Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Розробка веб-сервісу надання інформаційної підтримки цивільного населення під час надзвичайних ситуацій воєнного характеру»

Виконав: студент групи К19-2
Спеціальність 122 «Комп'ютерні науки»
Кожем'яка О. Л.
(прізвище та ініціали)

Керівник: к. ф.-м. н., доц. Лебідь О. Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент: Спеціалізоване управління розробки та супроводження програмного забезпечення Департамент з питань цифрового розвитку, цифрових трансформацій та цифровізації ДМСУ
(місце роботи)

головний державний інспектор відділу розробки програмного забезпечення
(посада)

Бахтін О. В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2023

АНОТАЦІЯ

Кожем'яка О. Л. Розробка WEB-сервісу надання інформаційної підтримки цивільному населенню під час дій надзвичайних ситуацій воєнного характеру.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 122 «Комп'ютерні науки» – Університет митної справи та фінансів, Дніпро, 2023.

Актуальність теми пов'язана з надзвичайними ситуаціями воєнного характеру, що спостерігаються на території України та розробкою веб-сервісу надання актуальної інформації для громадськості в швидкій взаємодії з волонтерським рухом, що забезпечить обізнаність суспільства.

Проблемним питанням є складність, неуніфікованість WEB-ресурсів, що надають доступ до необхідної інформації. Об'єкт дослідження – сучасні технології розробки веб-сервісів. Предмет дослідження – розробка веб-сервісу з використанням сучасних технологій і можливостей.

Метою роботи є розробка веб-сервісу з відкритим вихідним кодом для забезпечення доступу до останніх новин, взаємодії з волонтерами, отримання інформаційної допомоги з урахуванням потреб користувача.

Проблемне питання вирішене шляхом створення WEB-застосунку, що дозволить користувачам, використовуючи простий і зрозумілий інтерфейс, отримати швидкий доступ до необхідних даних. В результаті, це дозволить централізувати та спростити отримання важливих знань для користувачів.

Практичне значення веб-сервісу, розробленого в кваліфікаційній роботі, полягає в тому, що він дозволить суспільству використовувати систему, яка вирішить проблему з доступом до актуальної інформації та можливостей отримати інформаційну допомогу.

Структура кваліфікаційної роботи: вступ, 3 розділи, висновки, обсяг роботи – 55 сторінки, містить 48 рисунків, перелік використаних джерел налічує 44 посилань.

Ключові слова: WEB-СЕРВІС, ASP.NET, C#, ANGULAR, TYPESCRIPT, СЕРВЕР, ІНТЕРФЕЙС, CLEAN ARCHITECTURE.

ANNOTATION

Oleksii Kozhemiaka Development of a web service for providing information support to the civilian population during military emergencies.

Qualification work for a bachelor's degree in speciality 122 "Computer Science" - University of Customs and Finance, Dnipro, 2023.

The relevance of the topic is related to the military emergencies observed on the territory of Ukraine and the development of a web service to provide up-to-date information to the public in rapid interaction with the volunteer movement, which will ensure public awareness.

The problematic issue is the complexity and unification of web resources that provide access to the necessary information. Object of research - modern technologies of web services development. The subject of research is the development of a web service using modern technologies and capabilities.

The purpose of the study is to develop an open source web service to provide access to the latest news, interaction with volunteers, and information assistance based on the user's needs.

The problematic issue was solved by creating a web application that will allow users to get quick access to the necessary data using a simple and intuitive interface. As a result, it will centralise and simplify the acquisition of important knowledge for users.

The practical significance of the web service developed in the qualification work is that it will allow society to use a system that will solve the problem of access to up-to-date information and opportunities to receive information assistance.

The structure of the qualification work is as follows: introduction, 3 chapters, conclusions, the volume of work is 55 pages, contains 48 figures, the list of references includes 44 references.

Keywords: WEB SERVICE, ASP.NET, C#, ANGULAR, TYPESCRIPT, SERVER, INTERFACE, CLEAN ARCHITECTURE.

ЗМІСТ

ВСТУП

Терористична війна Росії проти України змінили життя мільйонів українців, велика частина жінок, літніх людей та дітей були вимушені покинути свої домівки на виїхати за межі країни у пошуках прихистку.

Велика частина населення вирішили залишитись на території держави через різні причини, такі як страх, перед новою країною, або небажання покидати рідну Батьківщину. Через це вони опинились у скрутному становищі, оскільки частина з них покидали свої домівки, лишаючи усе нажите в них. У той же час, Росія веде війну неконвенційними методами, скидаючи ракети на голови простих українців, обстрілюючи прифронтові міста з артилерії та знищуючи інфраструктуру держави. У такій ситуації актуальним є волонтерський рух, який завдяки підтримці небайдужих громадян, допомагає громадському населенню.

Окремою проблемою є інформаційна війна, через що населення може мати викривлене уявлення про сьогодення, що є критичним, особливо, на тимчасово окупованих територіях, на яких ворог намагається максимально задурити поневолене населення, задля зупинення супротиву проти окупантів. Таким чином вони намагаються асимілювати та розчинити українську ідентичність на захоплених територіях. Зупинити негативний вплив від перебування населення під окупацією можливо, лише використовуючи сучасні засоби інформування суспільства, створюючи авторитетні інформаційні ресурси, надаючи безперервний доступ до них.

Тема кваліфікаційної роботи актуальна, так як надає можливість для громадськості в швидкій взаємодії з волонтерським рухом, надання актуальної інформації, що забезпечить обізнаність суспільства, підтримає національну свідомість населення, унеможливить спокійне перебування окупантів на нашій території через активне інформування та нагадування, що окупація не назавжди та рано чи пізно вони повернуться під контроль країни.

Розв’язання поставленого завдання забезпечить створення потужного інформаційного сервісу, який зможе убезпечити населення від методів інформаційної війни, надасть підтримку в разі виникнення критичних ситуацій, тому значення для предметної області оцінюється як значуще.

Об’єкт дослідження: сучасні технології розробки веб-сервісів.

Предмет дослідження: розробка веб-сервісу з використанням сучасних технологій і можливостей.

Метою кваліфікаційної роботи є аналіз сучасних засобів розробки, архітектурних підходів, розробка веб-сервісу з відкритим вихідним кодом для забезпечення доступу до останніх новин, взаємодії з волонтерами, отримання інформаційної допомоги з урахуванням потреб користувача, створення простого та зрозумілого сервісу з докладною інструкцією користувача.

Завдання кваліфікаційної роботи: провести аналіз проблемного питання, що розглядається в роботі; сформулювати постановку завдання, порівняльному аналізі існуючих реалізацій, складання технічного завдання, розробка вимог, аналізі сучасних методів та засобів розробки, проєктування архітектури сервісів, використання сучасних архітектурних рішень, створенні веб-серверів, розміщенні вихідного коду у відкритих джерелах, надання доступу до ресурсів проєкту суспільству, створенні вичерпної інструкції для користувачів.

Аналіз сучасних технологій є важливим етапом в процесі проектування програмного забезпечення, який дозволяє обрати технології, які зможуть у повному обсязі реалізувати завдання. В процесі аналізу засобів розробки необхідно враховувати сумісність засобів з обраними технологіями та їх надійність.

Сучасний рівень розв’язання завдання характеризується як незадовільний, оскільки не було знайдено існуючої системи, яка у повному обсязі задовольняла завданню. В процесі аналізу було визначено декілька рішень, що відповідають темі диплома, але вони мають суттєві недоліки, що не дозволяють їм у повній мірі вирішити завдання.

Методи дослідження, що застосовуються в роботі: огляд літератури, аналіз вимог, проєктування та розробка, експериментальне тестування.

Результати продемонстровані в кваліфікаційній роботі відповідають наступним результатам навчання освітньої програми 122 «Комп'ютерні науки»:

ПР1. Застосовувати знання основних форм і законів абстрактно-логічного мислення, основ методології наукового пізнання, форм і методів вилучення, аналізу, обробки та синтезу інформації в предметній області комп'ютерних наук;

ПР9. Розробляти програмні моделі предметних середовищ, вибирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв'язання задач в галузі комп'ютерних наук,

ПР10. Використовувати інструментальні засоби розробки клієнт-серверних застосувань, проектувати концептуальні, логічні та фізичні моделі баз даних, розробляти та оптимізувати запити до них, створювати розподілені бази даних, сховища та вітрини даних, бази знань, у тому числі на хмарних сервісах, із застосуванням мов веб-програмування;

ПР11 Володіти навичками управління життєвим циклом програмного забезпечення, продуктів і сервісів інформаційних технологій відповідно до вимог і обмежень замовника, вміти розробляти проектну документацію (техніко-економічне обґрунтування, технічне завдання, бізнес-план, угоду, договір, контракт).

Практичне значення кваліфікаційної роботи - веб-сервіс розроблений в даній кваліфікаційній роботі, який реалізує мету, що дозволить суспільству використовувати систему, яка вирішить проблему з доступом до актуальної інформації та можливостей отримати інформаційну допомогу. Відкритість коду та вільний доступ до репозиторію дозволить зацікавити проектом інших розробників, які на волонтерських засадах зможуть підтримати проєкт, створенням та підтримкою його можливостей. Інструкція користувача, яка

дозволить полегшити процес інтеграції користувачів у нову систему, забезпечить швидкість та надійність використання.

Явного економічного ефекту від рішення завдання не передбачається, оскільки проєкт позиціонується як надбання громадськості, однак як можливість, створення умовного благодійного фонду на підтримку проєкту, у який усі бажаючі зможуть зробити пожертви.

Структура кваліфікаційної роботи: вступ, 3 розділі, висновки, перелік використаних джерел, 2 додатки. Обсяг роботи – 55 сторінок, робота містить 48 рисунків, перелік використаних джерел складається з 44 посилань, додатки розміщено на 17 сторінках.

РОЗДІЛ 1.
АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

У цьому розділі роботи ми ретельно проаналізуємо проблемну область, пов'язану з наданням інформаційної підтримки громадському населенню під час надзвичайних ситуацій воєнного характеру. Даний аналіз дозволить нам краще зрозуміти важливі аспекти цієї проблеми та визначити необхідні кроки для розробки ефективного веб-сервісу, допоможе нам отримати глибше розуміння важливості розробки такого сервісу.

1.1 Задача створення інформаційної Web-системи підтримки цивільного населення під час надзвичайних ситуацій воєнного характеру

Задача створення інформаційної Web-системи полягає у створенні веб-серверу, що зможе на запити користувача надсилати відповіді у вигляді HTML сторінок, або у вигляді JSON, XML – об’єктів. Для вирішення цієї проблеми існує багато рішень, ось деякі з них: Spring, PHP, ASP.NET. В цьому випадку для вирішення цієї задачі буде використовуватись ASP.NET Web Api і ASP.NET MVC. Перший дозволяє створювати Web-сервери, що у відповідь на запити надсилають JSON об’єкти. Другий – у відповідь надсилає HTML сторінки.

У разі, якщо сервер надає інформацію у вигляді JSON, XML – об’єктів, необхідно створити клієнтський додаток, який буде працюючи в браузері представляти інформацію користувачу, відповідати за комунікацію з серверами та інше. Для цього існують наступні рішення: Angular, Vue, React. Використаємо Angular для створення клієнтського додатку. Додатково, необхідно подбати про дизайн сторінок клієнтського застосунку. Для цього доречно буде використати Bootstrap і Angular Material. Останній має інтегрованість у Angular проєкт та дозволяє створювати складні стилізовані об’єкти на сторінці.

Окрім цього, необхідно створити сховище даних, що буде зберігати, дозволить змінювати та видаляти, та забезпечить безвідмовний доступ до інформації Web-серверів. Для цього існує багато рішення у вигляді реляційних баз даних, найпопулярніші з них: Oracle Sql, MsSQL, MySQL, Postgresql. Використаємо MsSQL, оскільки вона має гарну інтеграцію з платформами ASP.NET.

Для того, щоб до сервісу був доступ з онлайну, необхідно розмістити його на сервері, що має такі функції. Для цього існують наступні найпопулярніші рішення: AWS, Google Cloud, Azure. Використаємо Azure, оскільки розробкою цього сервісу, так само як і MsSQL, і ASP.NET займається Microsoft і ці сервіси між собою добре інтегровані і потребують мінімальної кількості додаткових налаштувань.

Технічне завдання відповідно теми роботи матиме наступний вигляд:

1. Функціональні вимоги: реалізувати систему реєстрації та авторизації користувачів; забезпечити можливість отримання актуальних новин, попереджень та інструкцій щодо надзвичайних ситуацій; забезпечити можливість пошуку та отримання інформації про доступні медичні заклади, постачання продуктів та інші необхідні ресурси; реалізувати можливість комунікації між користувачами, дозволяючи обмінюватись інформацією, ресурсами та підтримкою; забезпечити систему зворотного зв'язку, включаючи звітність про проблеми, запитання та пропозиції від користувачів.

2. Нефункціональні вимоги: забезпечити інтуїтивно зрозумілий та привабливий інтерфейс користувача; забезпечити швидкий доступ до інформації та стабільну роботу веб-сервісу навіть при високих навантаженнях; забезпечити безпеку та конфіденційність обробки та передачі інформації; розробити систему резервного копіювання та відновлення даних для забезпечення надійності та безперебійності роботи сервісу; забезпечити сумісність веб-сервісу з різними браузерами та пристроями.

3. Технічні вимоги: використання сучасних технологій веб-розробки, таких як HTML, CSS, JavaScript та фреймворки для створення динамічних веб-

додатків; застосування бази даних для зберігання інформації про користувачів, новини, ресурси та інші дані; забезпечення масштабованості та гнучкості архітектури, щоб забезпечити можливість розширення функціональності та підтримки більшої кількості користувачів.

1.2 Аналіз проблеми інформаційної Web-системи підтримки цивільного населення під час надзвичайних ситуацій воєнного характеру

Через війну Росії проти України, нашій державі завдані мільярдні економічні збитки, економіка функціонує лише завдяки вливанню в неї західних коштів, мільйони людей потребують уваги і допомоги.

Оскільки країна не була готова до таких подій і не має змоги швидко наладити таких масштабів систему допомоги постраждалим, волонтерство є необхідним і затребуваним видом діяльності.

Одна з проблем, з якою стикаються волонтери і ті, хто шукають допомогу – це швидкий, структурований доступ до актуальної інформації.

З однієї сторони волонтери, які мають бажання допомогти нужденним, але не мають відкритих і безкоштовних інформаційних систем для публікації своїх послуг. Звісно, вони користуються українськими телеграм каналами та іншими засобами масової інформації, однак й вони не справляються з кількістю волонтерів, оскільки не можуть дозволити собі цілий день публікувати лише волонтерські новини.

З іншої сторони – люди, що потрапили у неприємну життєву ситуацію. На жаль, це може статися у будь-який момент, тому шукати інформацію про волонтерський хаб в своєму місті в новині, яка була опублікована 3 дні тому в якійсь стрічці, дуже незручно. У такий ситуації людині необхідна швидка актуальна інформація або місце, де можна запитати про це.

Ідея створення інформаційної волонтерської системи з фокусом на доступ до актуальної інформації про волонтерство на території України є актуальною та затребуваною.

1.3 Порівняльний аналіз існуючих реалізацій проблеми

В процесі аналізу існуючих рішень було обрано 4 найкращих:

1. Платформа для швидкого пошуку волонтерської допомоги в усіх областях України https://palyanytsya.info/ - інформаційний портал для пошуку волонтерської допомоги. Налічує більше ніж 900 волонтерських організацій.

Дозволяє відфільтровувати наявні в базі даних волонтерські організації та представляє коротку інформацію про них, таку як опис діяльності та контакти (рисунок 1.1).

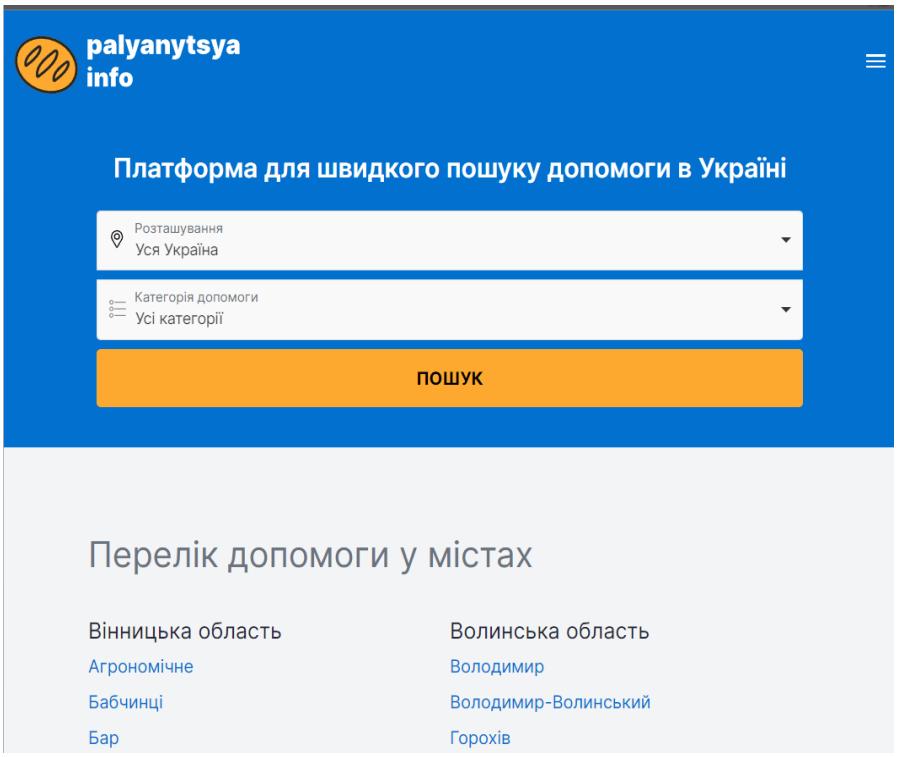


Рисунок 1.1 – Домашня сторінка платформи для швидкого пошуку допомоги в Україні

Оновлення бази даних відбувається за допомогою модераторів сервісу. Користувач може відправити запит на додавання організації у базу, після перевірки модератору вона додається у загальний список.

Основні переваги цього сервісу включають: структурованість та повнота інформації, вдалий пошук за населеним пунктом (рисунок 1.2).

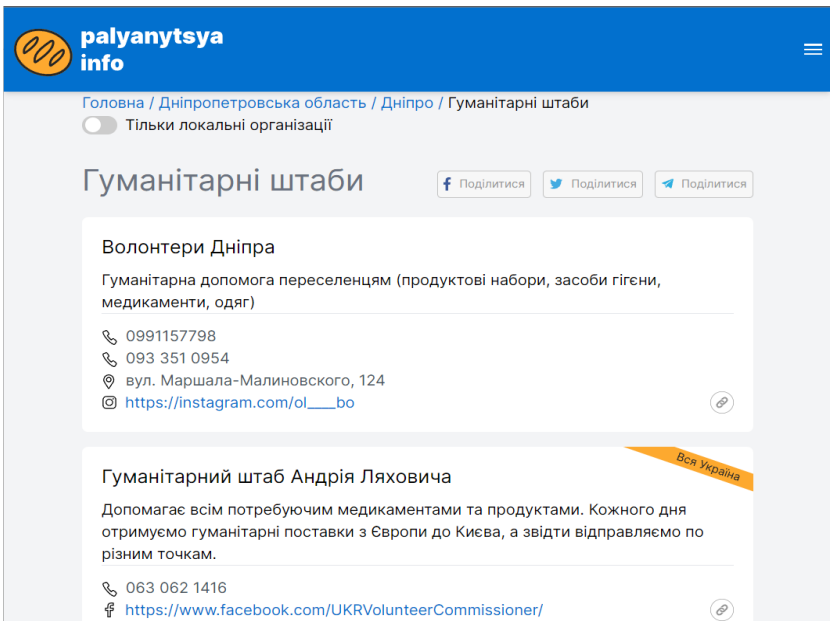


Рисунок 1.2 – Вигляд результатів пошуку гуманітарних штабів в місті Дніпро

Основні недоліки: складність в розумінні інформації – волонтерських хабів забагато, а їх опис не конкретний, загальний, не зрозуміло, чи зможе допомогти волонтерський хаб у конкретній ситуації.

2. Волонтерська платформа https://platforma.volunteer.country/ - інформаційний портал для волонтерів для пошуку однодумців і бажаючих бути волонтером (рисунок 1.3).

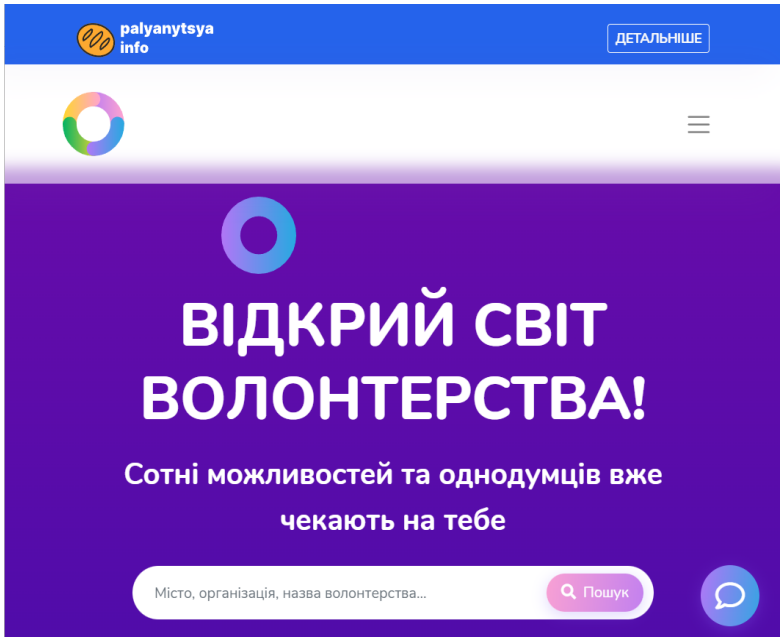


Рисунок 1.3 – Домашня сторінка волонтерської платформи

Дозволяє зареєструватись як волонтерська організація та публікувати повідомлення, щодо необхідної допомоги для волонтерського хабу (рисунок 1.4). Після реєстрації необхідно дочекатися підтвердження профілю від модераторів.

Основна перевага: реєстрація та валідація волонтерів, створення ними оголошень.

Основними недоліками є: відсутність вичерпної інформації про волонтерський хаб, складність в реєстрації, відсутність можливості розміщення інформації, окрім оголошень.

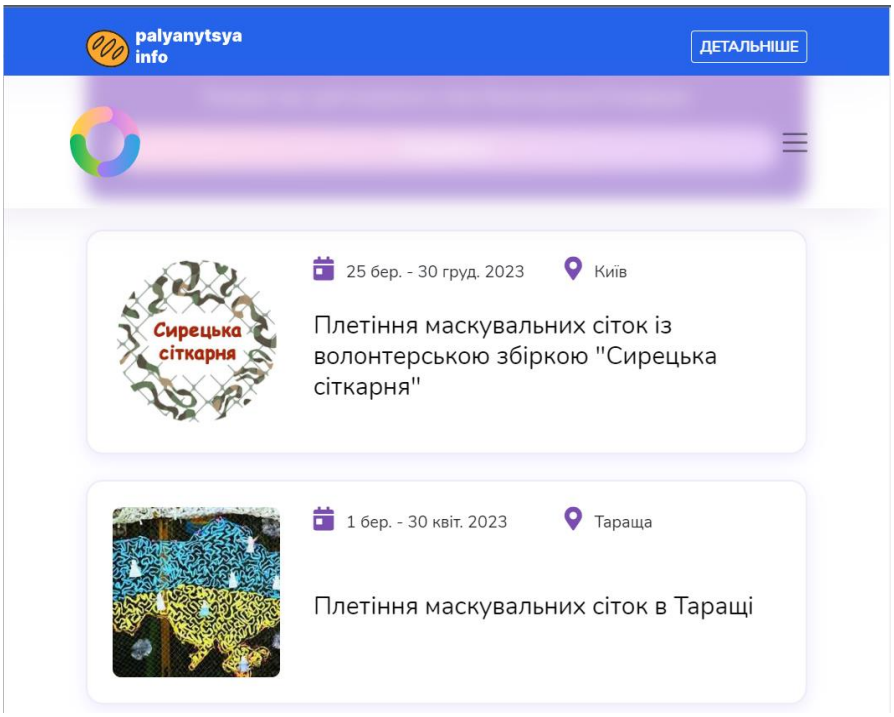


Рисунок 1.4 – Результати пошуку за запитом «Допомога військовим»

3. Незламність https://nezlamnist.gov.ua/ – інформаційний портал у вигляді інтерактивної мапи на якій розміщено пункти незламності (рисунок 1.5).

Дозволяє знаходити пункти незламності, що створені за підтримки держави та бізнесу. Користувач має змогу побачити на мапі місце знаходження найближчого пункту незламності, в якому він зможе отримати їжу, обігрітися або підзарядити телефон.

Основна перевага: розміщення на мапі є вдалим рішенням, легко знайти за своїм місцезнаходженням найближчий пункт.

Основний недолік: відсутність інформації, окрім пунктів незламності.

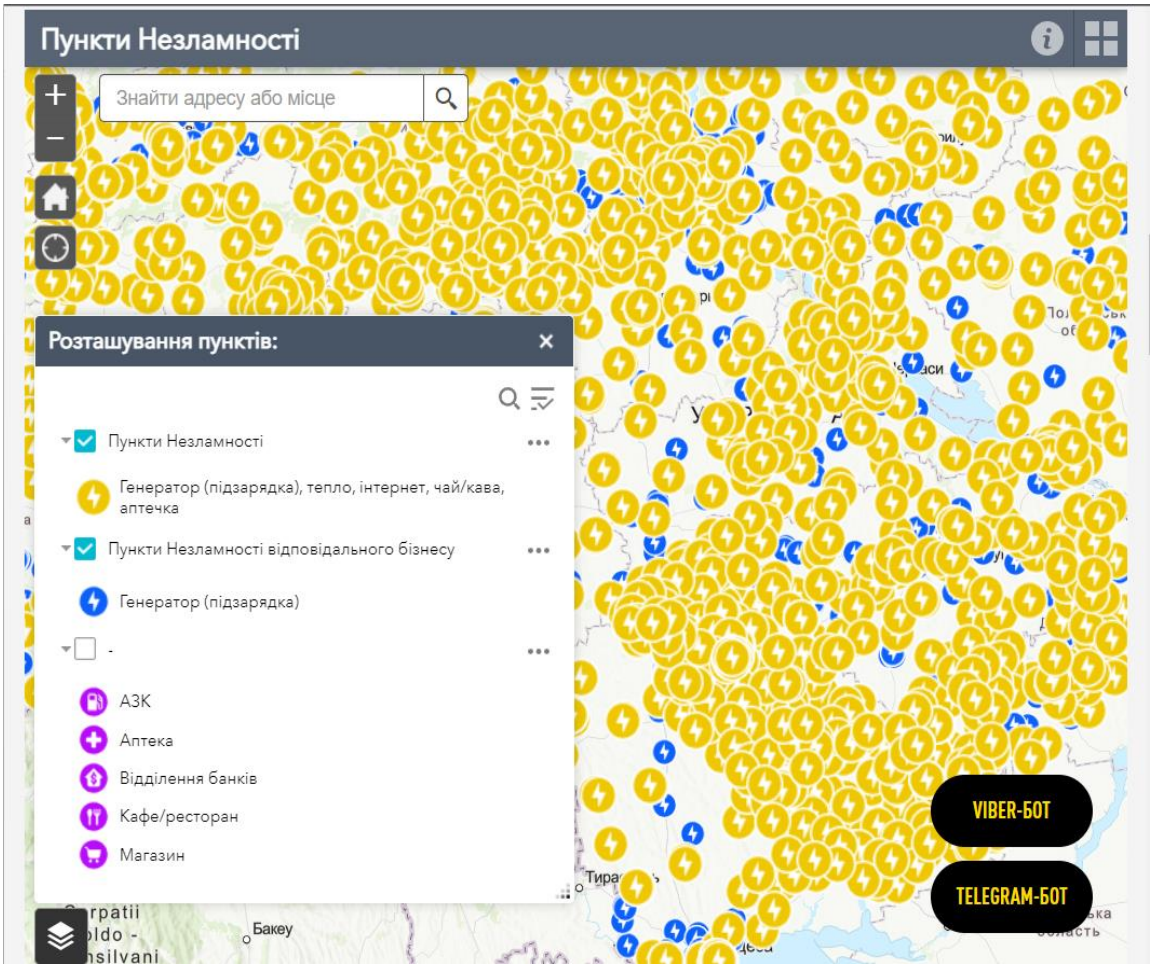


Рисунок 1.5 – Вигляд мапи незламності

4. Останні події країни https://www.ukr.net/news/main.html – останні головні новини країни. Іноді публікуються новини про волонтерські хаби.

Користувач отримує доступ до коротких основних новин у вигляді списку, та, за бажанням, має змогу переглянути повну новину (рисунок 1.6).

Основна перевага: присутня альтернативна інформація, що може бути корисною.

Основні недоліки: відсутність фільтрів і пошуку за новинами, неможливість відфільтрувати перегляд лише інформації про волонтерство, відсутність реєстру волонтерів.

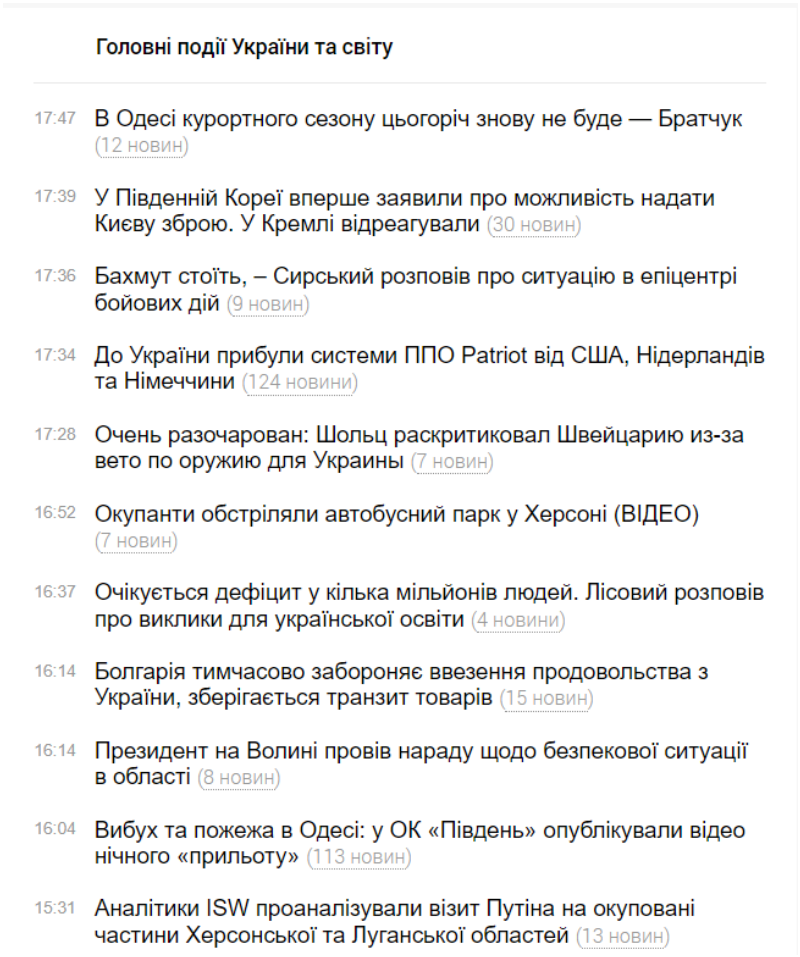


Рисунок 1.6 – Вигляд стрічки останніх новин

1.4 Висновки до першого розділу

Проведено аналіз проблемної області, розглянуто засоби рішення проблеми та проаналізовано схожі існуючи рішення. До розгляду представлено 4 схожих рішення для кожного підкреслено його недоліки та переваги над іншими. Сформульовану задачу, виконання якої забезпечить виконання завдання кваліфікаційної роботи.

РОЗДІЛ 2.
ЗАСОБИ РОЗРОБКИ

Для виконання завдання необхідно застосовувати різні засоби розробки, для зручності, можемо виділити три основні групи – Backend, Frontend технології та інші допоміжні бібліотеки та сервіси.

2.1 Backend технології

1. Мова програмування C#

C# — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (належить Microsoft).

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато від своїх попередників — мов C++, Object Pascal, Модула і Smalltalk — C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова C#, на відміну від C++, не передбачає множинне успадкування класів [1].

C# був розроблений компанією Microsoft, при створені автори надихались мовою програмування Java, що була дуже популярною у час розробки C# [2].

2. Платформа ASP.NET

ASP.NET — технологія створення веб-застосунків і веб-сервісів від компанії Майкрософт. Вона є складовою частиною платформи Microsoft.NET і розвитком старішої технології Microsoft ASP. На цей час останньою версією цієї технології є ASP.NET Core 6.0 [3].

Наразі, платформа ASP.NET є досить популярною та дозволяє швидко розгортати веб-додатки. Вона підтримує різні види ASP.NET, для реалізації задачі необхідно буде використати ASP.NET MVC та ASP.NET Web Api.

Перша технологія дозволяє створювати веб-додатки з використанням Razor pages engine, що необхідний при використанні представлень, для повернення HTML сторінок. Побудована на основі патерну MVC.

Друга – дозволяє створювати веб-додатки, що, як відповідь, повертають JSON об’єкти. Такий підхід дозволяє легко змінювати інтерфейс користувача, надаючи йому лише API свого серверу, та, дозволяючи йому самому вирішити, як опрацювати отриману інформацію.

3. Середовище розробки JetBrains Rider.

JetBrains Rider – IDE для розробки додатків з використанням .NET платформи. Платна, але для студентів існують програми, щодо отримання доступу до основних продуктів компанії JetBrains на безоплатній основі.

Rider – це потужна IDE, що дозволяє розробляти програми на .NET різних категорій: десктопні, веб, мобільні, тощо. [4] Користується популярністю через свою простоту, легкість у порівнянні з єдиною альтернативою Visual Studio, покращеним Intellisense. Інтерфейс користувача нагадує Visual Studio (рисунок 2.1).

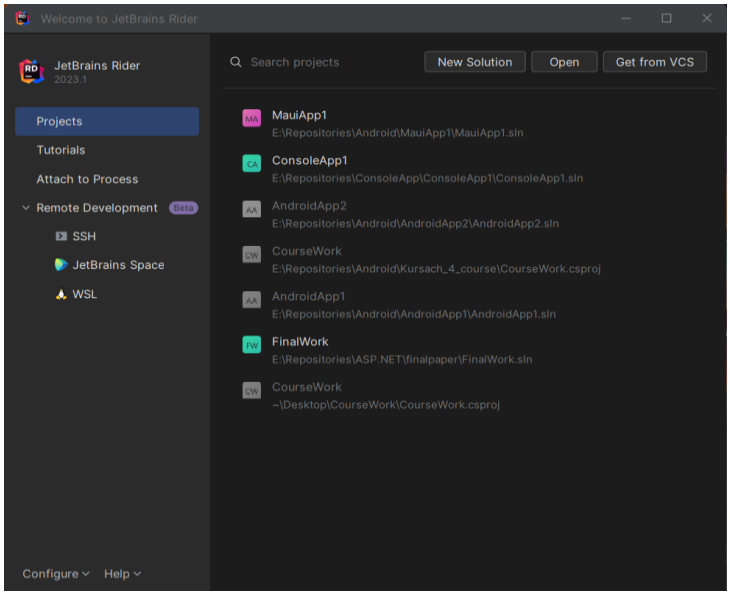


Рисунок 2.1 – Домашня сторінка Rider

Ліва частина вікна відведена під меню, а у правій, більшій частині, наведена історія останніх проєктів, що були запущені через Rider.

Таким чином, це дозволяє швидко повертатись до минулих проєктів та продовжувати роботу.

Створення нового проєкту наведено на рисунку 2.2.

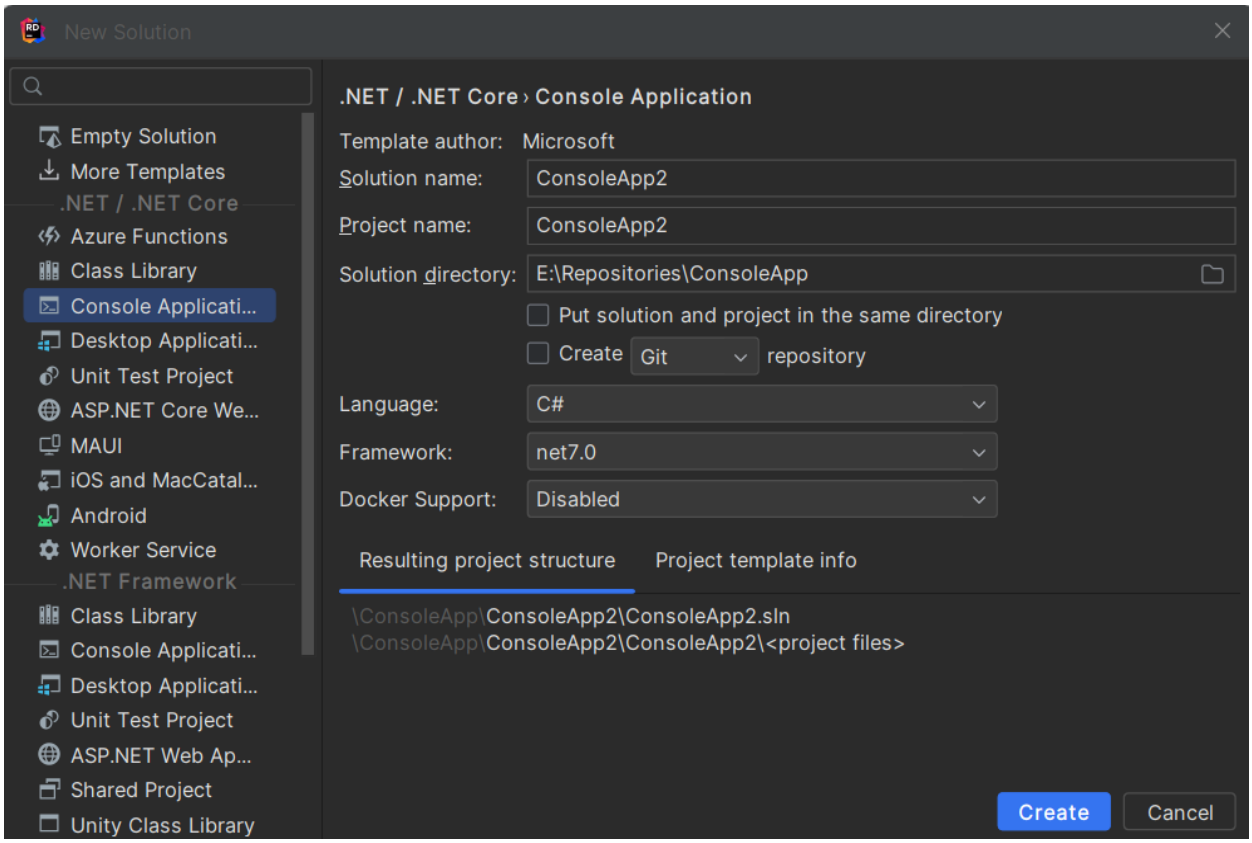


Рисунок 2.2 – Сторінка створення нового проєкту

Як видно, він дозволяє створювати різні .NET застосунки, за готовими шаблонами, окрім цього, підтримує застарілу технологію .NET Framework.

Робоче вікно в Rider містить багато різних меню, основна область виділена під код, є змога звернути меню, у нижній частині екрану додаткова функціональна область, що виділена під меню Git, NuGet Package Manager, консоль, тощо (рисунок 2.3).

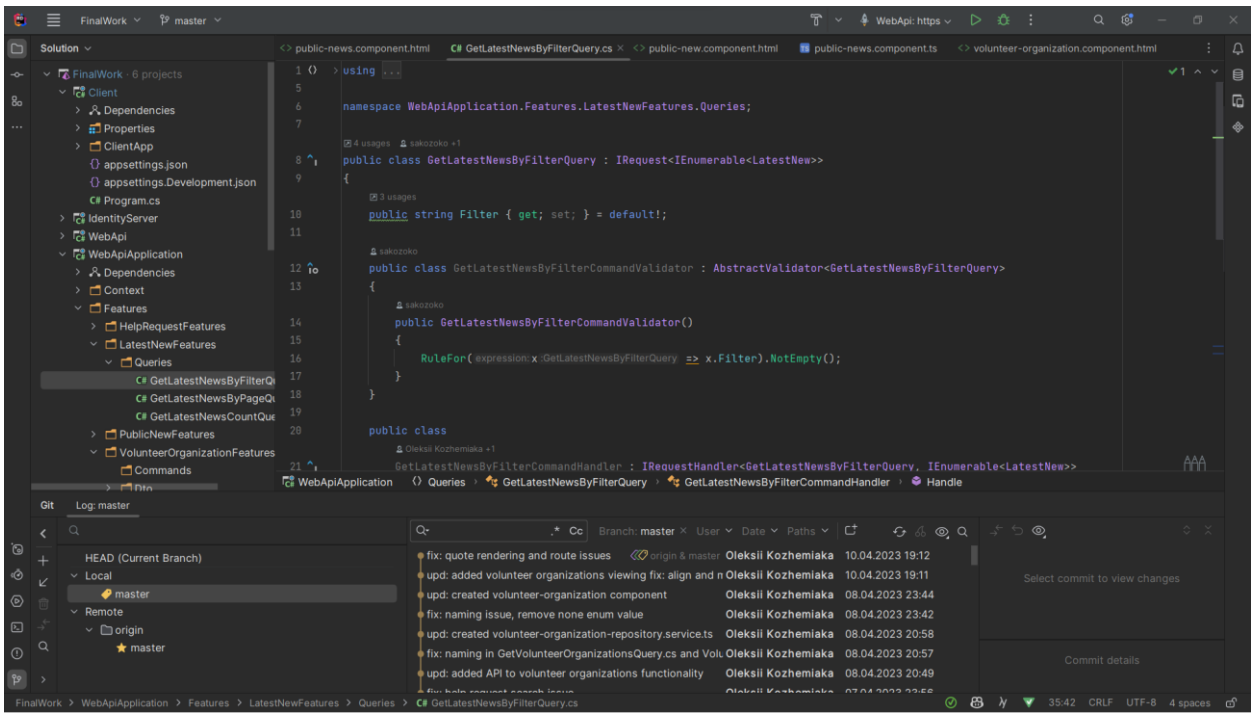


Рисунок 2.3 – Робоче вікно Rider

2.2 Frontend технології

1. Розмітка сторінок. HTML і CSS

HTML – стандартизована мова розмітки документів для перегляду веб-сторінок у браузері. Браузери отримують HTML документ від сервера за протоколами HTTP/HTTPS або відкривають з локального диска, далі інтерпретують код в інтерфейс, який відображатиметься на екрані монітора [3].

CSS – це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі ж сторінки написані мовами розмітки даних.

Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів.

CSS має різні рівні та профілі. Наступний рівень CSS створюється на основі попередніх, додаючи нову функціональність або розширюючи вже наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3. Профілі —

сукупність правил CSS одного або більше рівнів, створені для окремих типів пристроїв або інтерфейсів [5].

2. Bootstrap

Bootstrap – це безкоштовний CSS-фреймворк з відкритим вихідним кодом, призначений для адаптивної веб-розробки інтерфейсів, орієнтованих на мобільні пристрої. Він містить шаблони дизайну на основі HTML, CSS і (за бажанням) JavaScript для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу (рисунок 2.4) [6].

Він буде використаний для швидкого стилізування та створення адаптивності на сторінках. Дозволить зменшити написання CSS коду та створити спільний стильовий фон на усіх сторінках сервісів.

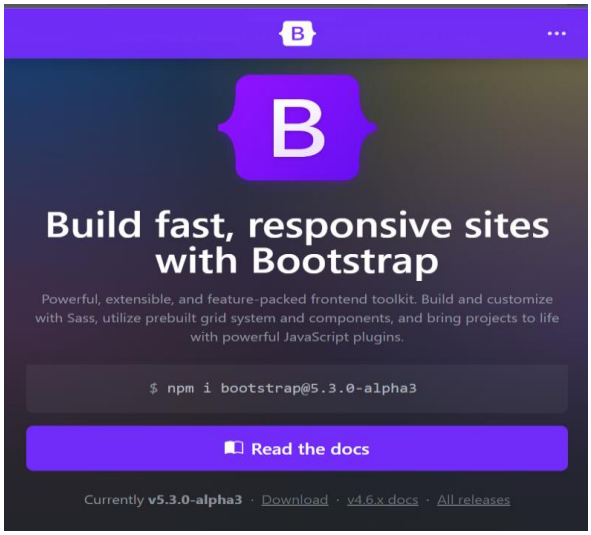


Рисунок 2.4 – Початкова сторінка документації Bootstrap

Як видно з документації, встановлення Bootstrap в проєкт не потребує багатьох зусиль.

3. Мова програмування TypeScript

TypeScript – мова програмування, що створена компанією Microsoft та представлена у 2012 році. Представляє з себе доповнення мови програмування JavaScript, але при цьому має статичну типізацію, підтримує використання класів та підключення модулів.

Користується великим інтересом у розробників, основною мовою яких є статично типізовані мови, такі як Java та C# [7].

З використанням цієї мови розроблена платформа Angular.

4. Платформа Angular

Angular – фреймворк розроблений компанією Google для створення клієнтських веб-додатків. Спочатку був створений на JavaScript, після чого переписаний з використанням TypeScript [8]. Є популярним серед .NET розробників, через використання TypeScript, хоча наразі і втрачає популярність на користь React. У проєкті буде використовуватись версія 15.2.4.

Angular має свою CLI, що працює на основі Node.JS, та дозволяє легко виконувати компіляцію проєкту, додавання нових компонентів, тощо [9].

Angular також має гарну документацію, що дозволить вирішити більшість питань, що можуть з’являтись у процесі використання цієї платформи (рисунок 2.5).

Рисунок 2.5 – Документація Angular. Створення першого проєкту Angular

2.3 Допоміжні бібліотеки та сервіси

Наведемо основні відомості по допоміжним бібліотекам та сервісам.

1. SQL, Microsoft SQL Server

SQL – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. Сама по собі SQL не є ані системою керування базами даних, ані окремим програмним продуктом. На відміну від дійсних мов програмування (C або Pascal), SQL може формувати інтерактивні запити або, бувши вбудованою в прикладні програми, виступати як інструкції для керування даними. Окрім цього, стандарт SQL містить функції для визначення зміни, перевірки та захисту даних.

SQL — це діалогова мова програмування для здійснення запиту і внесення змін до бази даних, а також керування базами даних. Багато баз даних підтримує SQL з розширеннями до стандартної мови. Ядро SQL формує командна мова, яка дозволяє здійснювати пошук, вставку, оновлення і вилучення даних за допомогою використання системи керування і адміністративних функцій. SQL також включає CLI (Call Level Interface) для доступу і керування базами даних дистанційно [10].

Microsoft SQL Server - система управління базами даних, яка розробляється корпорацією Microsoft. Як сервер даних виконує головну функцію по збереженню та наданню даних у відповідь на запити інших застосунків, які можуть виконуватися як на сервері, так і у мережі.

Мова, що використовується для запитів — Transact-SQL, створена спільно Microsoft та Sybase. Transact-SQL є реалізацією стандарту ANSI / ISO щодо структурованої мови запитів SQL із розширеннями. Використовується як для невеликих і середніх за розміром баз даних, так і для великих баз даних масштабу підприємства. Багато років вдало конкурує з іншими системами керування базами даних [10].

2. Microsoft Azure

Microsoft Azure – хмарний сервіс призначений для створення онлайнових додатків. Інтегрований в Visual Studio. Дозволяє легко розгортати веб-додатки на серверах Microsoft. Легко масштабується, за необхідністю, можна у будь-який час розширити використання сервісів. Платний, але надає для новостворених аккаунтів пільгові умови використання, що дозволяють 12 місяців не оплачувати основні сервіси на мінімальній продуктивності. Ці сервіси включають наступні, що необхідні будуть для розробки: Web app service, SQL server, SQL database, Key vault service.

Web app service – ресурсу Azure, що дозволяє створювати сервер, на якому зможе працювати веб-додаток. Для проєкту необхідні будуть три Web app service, два з яких з підтримкою ASP.NET, а один Node.JS.

SQL server/database – Microsoft SQL Server, що буде працювати у хмарі та до якого буде доступ з веб-додатків.

Key vault service – сервіс, що необхідний для збереження чутливої інформації, такої як: рядки підключення, ключі автентифікація на сторонніх сервісах, паролі, тощо. Дозволяє безпечно зберігати чутливу інформацію та надає доступ лише в межах Azure сервісам, які завчасно визначені в налаштуваннях ресурсу Key vault.

Домашню сторінку Azure після авторизації наведено на рисунку 2.6.

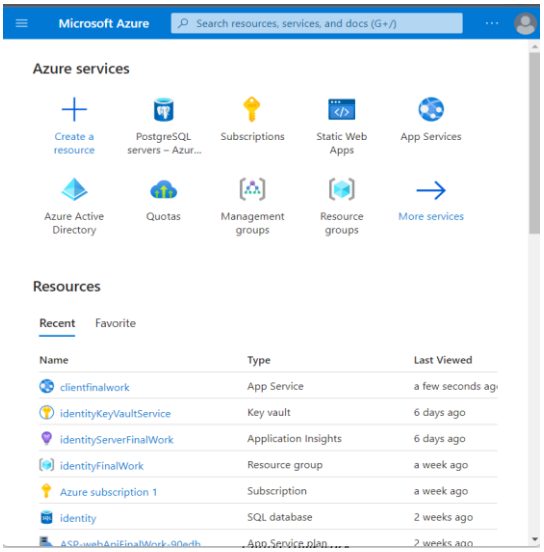


Рисунок 2.6 – Домашня сторінка Azure

Як видно, маємо пропозицію створити новий ресурс, шаблони ресурсів, та останні переглянуті раніше створені ресурси. При натисканні на ресурс відкривається сторінка з можливістю редагування налаштувань ресурсу (рисунок 2.7).

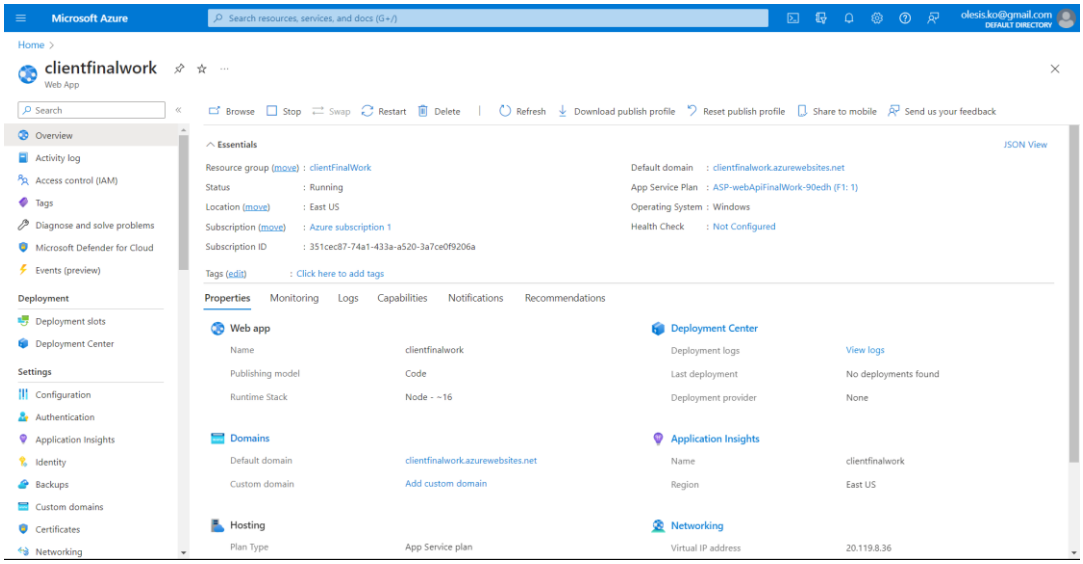


Рисунок 2.7 – Сторінка налаштування ресурсу

3. GitHub

GitHub – веб-сервіс для спільної розробки програмного забезпечення та базується на системі керування версіями Git.

Git – розподілена система керування версіями файлів та спільної роботи. Проєкт створив Лінус Торвальдс для керування розробкою ядра Linux, а сьогодні підтримується Джуніо Хамано (англ. Junio C. Hamano).

Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок [8].

Для зручнішої розробки проєкту буде створено відкритий репозиторій, в якому буде зберігатись вихідний код проєкту, та який буде доступний усім бажаючим. При створенні репозиторію необхідно ввести його ім’я та обрати його видимість публічний або приватний (рисунок 2.8).

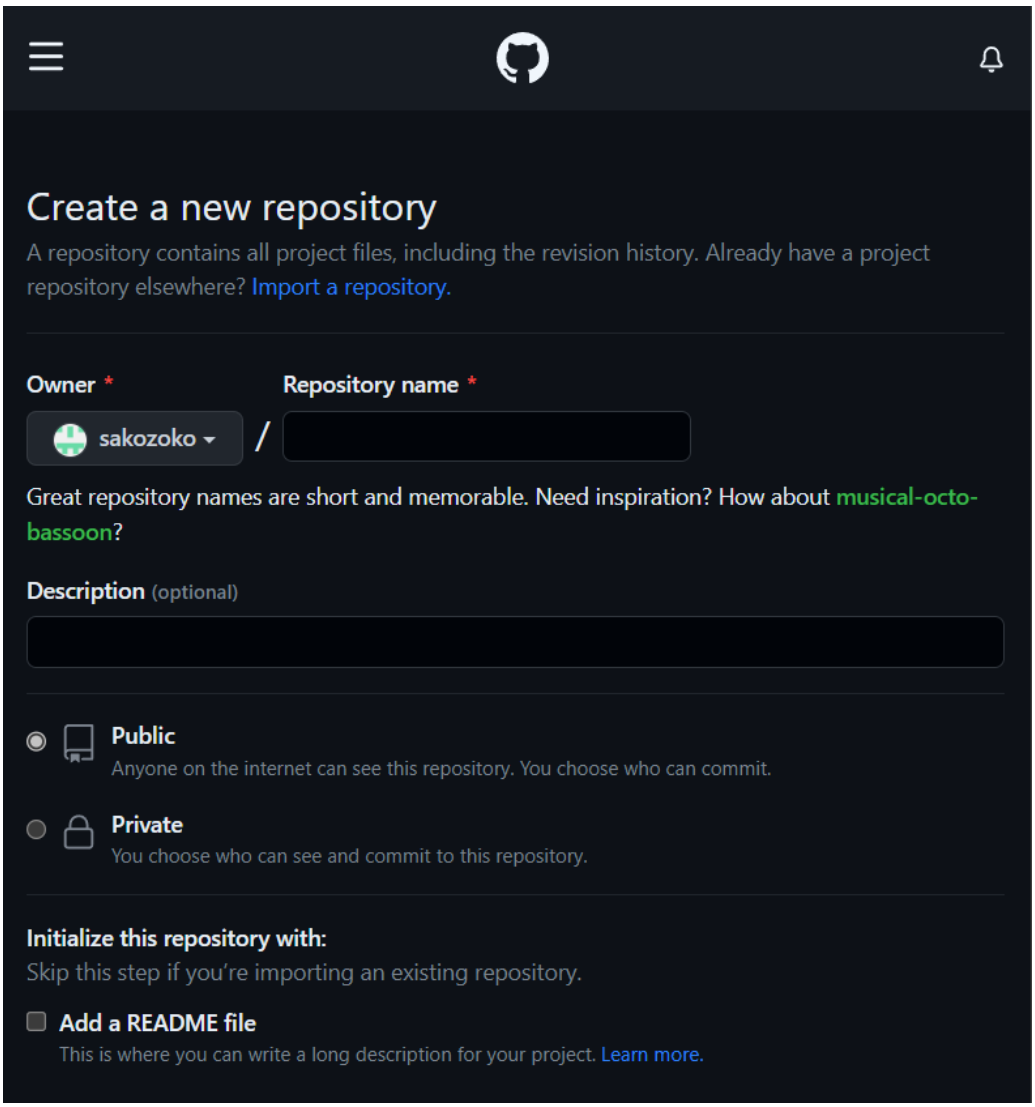


Рисунок 2.9 – Створення нового репозиторію GitHub

4. Бібліотека Entity Framework

Entity Framework (EF) - це об'єктно-реляційний маппер, який дозволяє .NET розробникам працювати з реляційними даними, використовуючи специфічні для домену об'єкти. Він усуває необхідність у більшості коду доступу до даних, який зазвичай доводиться писати розробникам [11].

Дозволить спростити доступ до даних. Для розробки використаємо підхід Code First, що дозволить створити спочатку моделі у вигляді C# коду, які пізніше EF перетворить у таблиці у Microsoft SQL Database.

5. Бібліотека Duende IdentityServer

Duende IdentityServer – сервер автентифікації, що дозволяє швидко і легко під’єднати автентифікацію у проєкт.

Буде реалізований у вигляді окремого серверу, який буде відповідати за авторизацію клієнтів, використовуючи JWT токени. Окрім цього, на нього будуть покладені завдання з реєстрації та модифікації профілю користувача, таким чином, усі дії з користувачами сервісів будуть покладені на 1 сервер.

Буде використана 6 версія бібліотеки. Підтримує взаємодію з ASP.NET Identity.

6. Бібліотека ASP.NET Identity

ASP.NET Identity – бібліотека, що інтегрована з Duende IdentityServer, та дозволяє легко створити свою модель користувача, налаштувати авторизацію, тощо. Бібліотека розроблена компанією Microsoft, має свою власну документацію.

7. Бібліотека AutoMapper

AutoMapper – бібліотека, що дозволяє легко мапити одні моделі до інших. Необхідна, оскільки проєкт буде мати багаторівневу архітектуру, та кожен рівень буде працювати зі своїми моделями. Окрім цього, він гарно інтегрується з іншими бібліотеками, такими як: MediatR та FluentValidation.

8. Бібліотека MediatR

MediatR – реалізація патерну посередник. Гарно поєднується з патерном Command and Query Responsibility Segregation, що буде використаний в реалізації веб-сервісів.

9. Бібліотека FluentValidation

FluentValidation – бібліотека, що зможе полегшити валідацію запитів користувача до серверу, окрім цього, дозволить зберігати увесь код, що пов’язаний з командою у одному файлі, не порушуючи принципів SOLID.

10. Бібліотека Azure.Security.KeyVault.Secrets

Azure.Security.KeyVault.Secrets – бібліотека, що полегшує взаємодії з Azure Key vault service. Дозволяє легко отримувати значення секретів за отриманим ключем. Необхідна для отримання чутливої інформації.

11. Бібліотека і сервіс SendGrid

SendGrid – бібліотека, що полегшує взаємодію з сервісом SendGrid.

SendGrid – сервіс, що дозволяє легко і безпечно розсилати email повідомлення, буде використаний для нотифікація користувача на його електронну пошту. Сервіс створено командою розробників сервісу Twilio, використання одного аккаунту в цьому сервісі дозволяє використовувати і безпечні email повідомлення і легко реалізувати SMS нотифікацію.

12. Бібліотека і сервіс Twilio

Twilio – бібліотека для зручної взаємодії з сервісом Twilio.

Twilio – сервіс, що дозволяє пересилати повідомлення на мобільний телефон користувача. Сервіс є досить популярним та займає велику нішу зі створення та поставки SMS повідомлень по всьому світу. Має велику документацію, дозволяє інтегрування з такими мовами, як C#, PHP, Java та інші.

13. Бібліотека HtmlAgilityPack

HtmlAgilityPack – бібліотека, що дозволяє зручно парсити html-сторінки, що буде необхідно для проєкту під час отримання деяких даних з інших сервісів. Використання цієї бібліотеки зумовлено якістю її реалізації, доступності документації та відкритості коду у репозиторії на GitHub, що дозволить у разі виникнення проблем звернутися до репозиторію за необхідної інформації, щодо використання бібліотеки.

14. Сервіс Abstract API

Abstract API – сервіс, що, окрім іншого, дозволяє валідувати номера телефонів, та повертає їх інтернаціональний формат. Це дозволить перекласти відповідальність за валідацію номерів телефонів на сторонній сервіс та отримувати стандартизований формат мобільних телефонів замість того, що буде вводити користувач.

15. Бібліотека Intl-Tel-Input

Intl-Tel-Input – бібліотека, що представляє зручний об’єкт вводу номеру, який доступний для користувача. Надає зручний контрол для користувачів, який забезпечить легкий ввід та пришвидшену валідацію на стороні клієнта номерів телефону.

16. Бібліотека Angular-oauth2-oidc

Angular-oauth2-oidc – бібліотека, що впроваджує легку взаємодію за протоколом OIDC та дозволить полегшити роботу для клієнтського додатку з сервером автентифікації. Ця бібліотека обрана серед інших через те, що має найбільш вичерпну документацію та прикладів використання та потребує мінімальних витрат для інтеграції з Angular.

17. Бібліотека Angular Material

Angular Material – бібліотека для стилізації Angular додатків. Має великий набір складних елементів, що дозволяють створювати інтерактивні додатки. Має інтегрованість в проєкт Angular, що в цьому випадку не актуальна, оскільки основний CSS Framework в проєкті – це Bootstrap. Бібліотека має велику популярність, що забезпечить швидке вирішення проблем, пов’язаних з її застосуванням.

2.4 Висновки до другого розділу

Розглянуто засоби розробки, що будуть використовуватись при створенні рішення дипломної роботи. Така кількість різних сервісів і бібліотек необхідна, оскільки дозволить виконати роботу якісніше та швидше, використовуючи менше ресурсів. Були розглянуті такі засоби: C#, MsSQL Server, ASP.NET, TypeScript, Angular, GitHub та інші. З найбільших та основних виділяється ASP.NET платформа та MsSQL Server, що потребують найбільшої підготовки перед використанням.

РОЗДІЛ 3.
РОЗРОБКА ІНФОРМАЦІЙНОЇ WEB-СЕРВІСУ

Цей розділ детально розкриває процес розробки інформаційної web-сервісу, включаючи архітектуру системи, інтерфейс користувача та методику роботи з програмним продуктом. Ця інформація є важливою для розуміння функціональності та можливостей веб-сервісу, який забезпечує ефективну інформаційну підтримку громадського населення під час надзвичайних ситуацій воєнного характеру.

3.1 Опис програмної реалізації. Архітектура системи

Інформаційна система передбачає реалізацію декількох сервісів, що необхідні для її роботи. Така структура дозволить легко редагувати різні проєкти, не порушуючи роботи інших. З цією метою створено 3 проєкти: сервер автентифікації, сервер Web Api та клієнтський додаток. Таким чином – це дозволить полегшити управління сервісами, зменшити навантаження на сервер та додасть гнучкості у розробці.

1. Сервер автентифікації.

Розроблений з використанням Duende IdentityServer та Microsoft.AspNet.Identity бібліотек. На сервер автентифікація покладаються наступні обов’язки:

1) Створення користувачів.
2) Авторизація користувачів.
3) Реалізація протоколу OIDC.
4) Представлення інформації профілю та додаткових можливостей:
   а) підтвердження номеру телефону;
   б) підтвердження пошти;
   в) зміна номеру телефону;
   г) зміна пошти.

5) Можливість відновлення паролю за допомогою надсилання на пошту листа-відновлення паролю.

Це мінімальний інтерфейс, що повинен забезпечувати сервер автентифікації. Такі вимоги дозволять у майбутньому легко масштабувати сервер, або змінити його зовсім.

Можливості Duende IdentityServer дозволяють авторизувати одночасно декілька сервісів, що в свою чергу, дозволить у майбутньому використовувати єдиний сервер автентифікації і один профіль на ньому для авторизації на декількох сервісах.

Оскільки сервіс не передбачає великих масштабів, то при його розробці використаний монолітний підхід до архітектурного рішення, що дозволить зменшити часові витрати на його розробку.

Структуру проєкту наведено на рисунку 3.1.

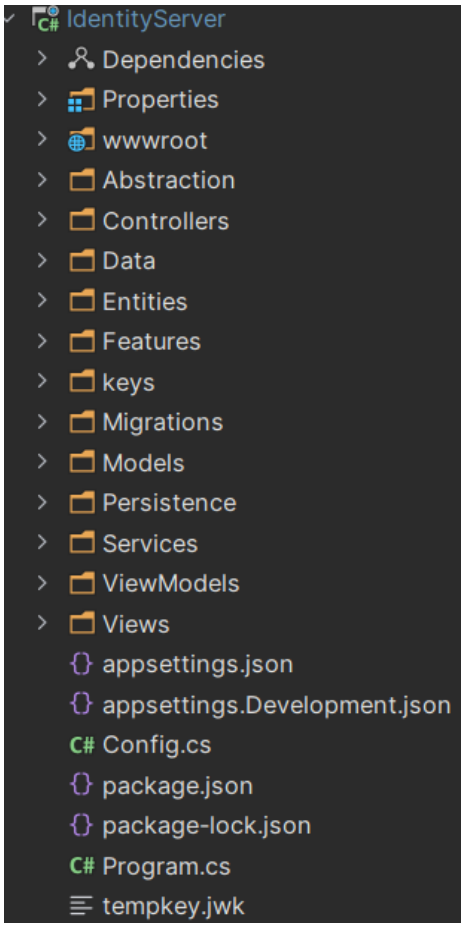


Рисунок 3.1 – Структура сервера автентифікації

2. Сервер Web Api.

Головний сервіс, що буде відповідати за всю взаємодію з об’єктами системи. Цей сервер передбачає наявність усього функціоналу, що буде доступний в клієнтському додатку. Робота з базою даних, валідація запитів, створення додаткових можливостей – задач покладено багато, саме тому при його реалізації використовувалась чиста архітектура (рисунок 3.2), що передбачає розбиття проєкту на декілька шарів, основні з який:

1) Domain;
2) Application;
3) Infrastructure;
4) Presentation;
5) Persistence.

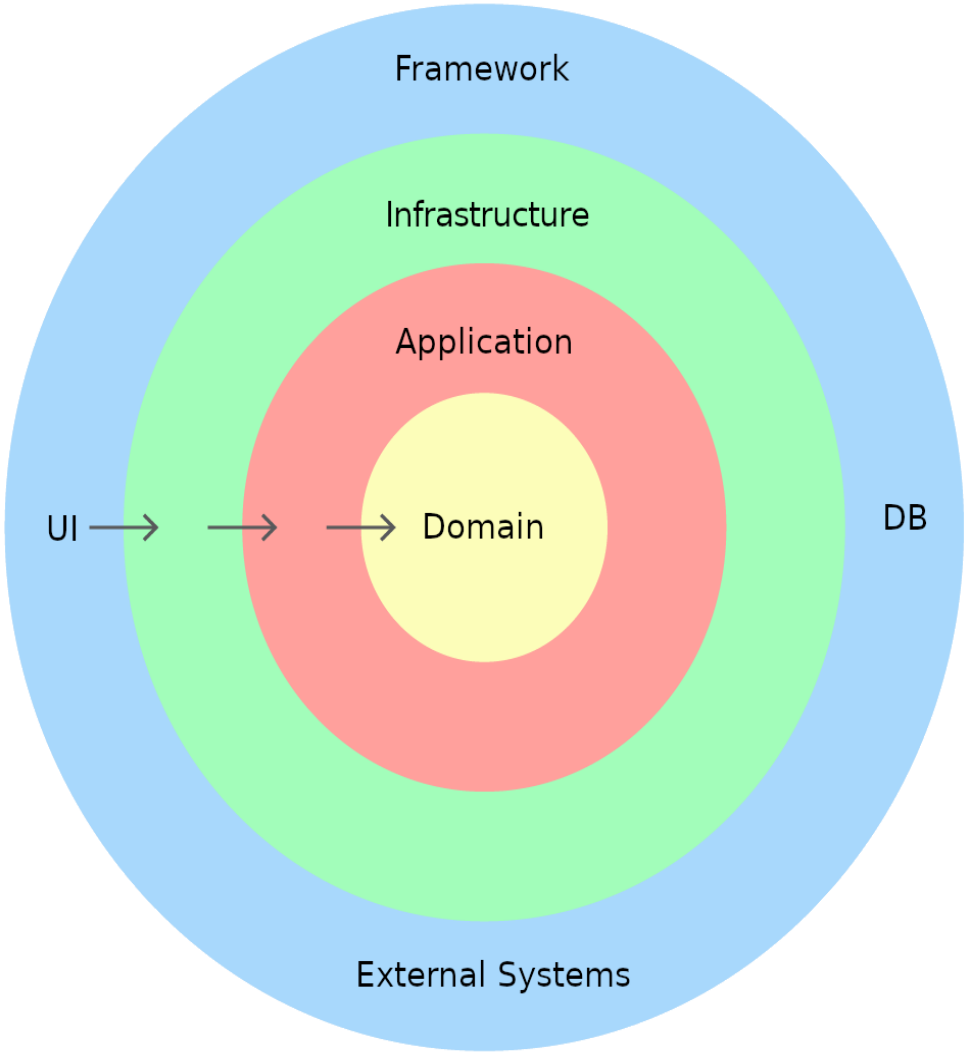


Рисунок 3.3 – Clean architecture

Основною рисою цієї архітектури є залежність зовнішніх шарів на внутрішні. Тобто, чим глибше знаходиться шар, тим більш незалежним він є.

У середину покладено шар з бізнес-моделями проєкту. Вони є найбільш незмінними в проєкті і тому, на них можуть посилатися інші шари.

Наступний шар – Application. В ньому зберігається увесь функціонал проєкту. Цей шар повинен посилатися лише на Domain, та бути незалежним від бази даних або клієнтського інтерфейсу. Також, тут повинні зберігатись інтерфейси доступу до даних. Таким чином, хоч і їх реалізація буде знаходитись рівнем вище, але основний принцип при цьому порушеним не буде. Це забезпечить можливість зміни сховища даних, а подробиці реалізації інкапсулює, оскільки вони безпосередньо не впливають на роботу додатку.

Окрім цього, для розділення відповідальності використовується патерн посередник. Його імплементацією є бібліотека MediatR. Також, для чистого використання методів, в цьому шарі задіяний патерн CQRS, що дозволяє розділити функції шару на команди та запити, де перші можуть виконувати операції вставки, оновлення та видалення, а другі – операції виведення інформації. В результаті, для кожної дії матимемо окремий клас, що дозволить застосувати один з принципів SOLID, а саме single responsibility principle, який передбачає, що в кожного класу має бути лише одна відповідальність.

```
public class GetHelpRequestsByFilterQuery : IRequest<IEnumerable<HelpRequestDto>>
{
    4 usages
    public string? Filter { get; set; }
    3 usages
    public Guid? UserId { get; set; }
    5 usages
    public int? Page { get; set; }
    6 usages
    public int? PageSize { get; set; }

    sakozoko
    public class GetHelpRequestsByFilterQueryValidator : AbstractValidator<GetHelpRequestsByFilterQuery>{...}

    sakozoko +1 *
    public class GetHelpRequestsByFilterQueryHandler : IRequestHandler<GetHelpRequestsByFilterQuery, IEnumerable<HelpRequestDto>>{...}
}
```

Рисунок 3.3 – Реалізація GetHelpRequestsByFilterQuery

Також, для валідації запитів використана бібліотека FluentValidation, яка дозволить легко реалізувати валідацію запитів.

Відповідно, як варіант імплементації одного з таких методів, наведемо приклад на рисунку 3.3.

Як видно, кожний клас запиту або команди матиме в собі наступні елементи:

1) безпосередньо запит – властивості, що будуть надходити з http-запитів.

2) валідатор – клас, в якому реалізована валідація параметрів запиту.

3) хендлер – клас, що займається виконанням запиту.

Причому валідація відбувається саме між створенням запиту та його виконанням, що не дозволить виконатись невалідному запиту.

Команди та запити наведені на прикладі HelpRequest моделі у додатку Б, інші виглядають типово.

Як результат, структура проєкту Application матиме вигляд, як на рисунку 3.4.

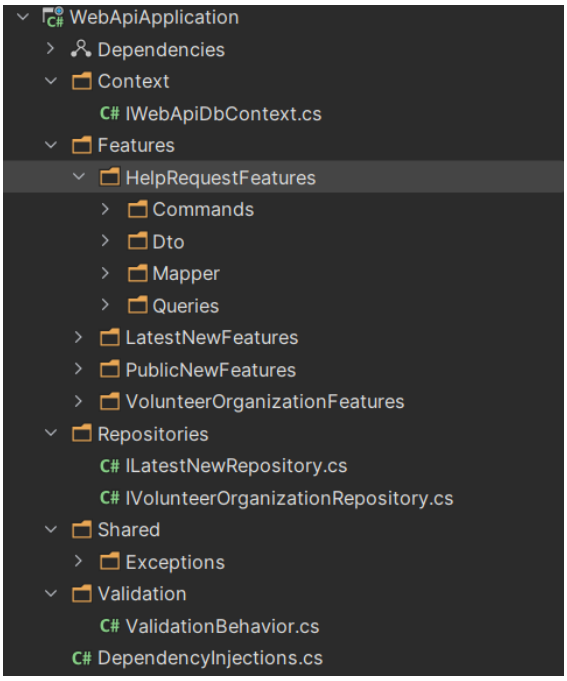


Рисунок 3.4 – Структура проєкту Web Api

Далі – Infrastructure. Шар, що буде зберігати в собі реалізацію роботи с базою даних. Він буде реалізовувати інтерфейси, що необхідні для роботи Application шару. Для реалізації цього шару використана бібліотека Entity

Framework. Окрім цього, тут також зберігаються реалізації репозиторіїв, що необхідні для роботи з сторонніми сервісами, що використовуються в роботі додатку. Структура проєкту Infrastructure наведена на рисунку 3.5.

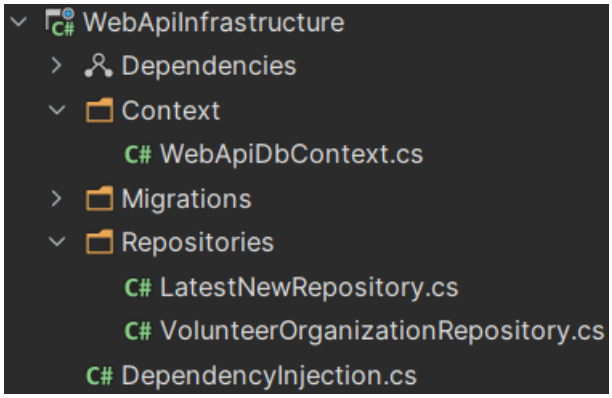


Рисунок 3.5 – Структура проєкту Infrastructure

Останній шар – Presentation. Тут знаходиться проєкт, що репрезентує дані, надає можливість до виконання операцій з методу Application, тощо. В нашому випадку це проєкт ASP.NET Web Api. Він має декілька контролерів, що надають доступ за HTTP запитами до можливостей сервісу. Оскільки вище ми використали патерн посередник, то реалізація методів контролерів виглядає як на рисунку 3.6.

```
[HttpGet( template: "availableCities")]
Oleksii Kozhemiaka
public async Task<IActionResult> GetAvailableCities([FromQuery] GetAvailableCitiesQuery command)
{
    return Ok(await _mediator.Send(command));
}
```

Рисунок 3.6 – Типова реалізація методу контролера

Тобто, ми отримуємо або створюємо модель запиту та надсилаємо його до посередника, той, в свою чергу, самостійно знаходить необхідний хендлер та оброблює запит, в результаті маємо мінімальну реалізацію, необхідну для контролерів. Лістинг контролерів наведено у додатку Б.

Структура проєкту наведена на рисунку 3.7.

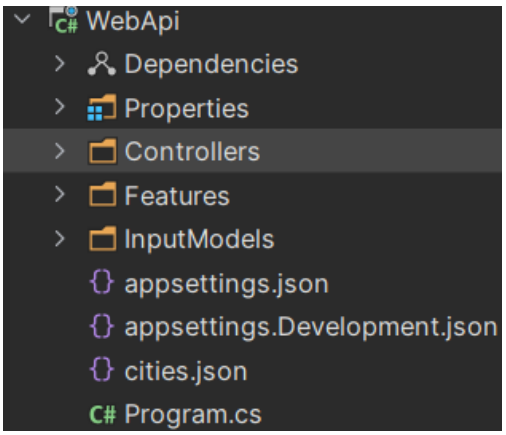


Рисунок 3.7 – Структура проєкту Web Api

Отже, завдяки використаним принципам і патернам, маємо гнучкий проєкт, що дозволяє легко створювати новий функціонал з мінімальної кількістю змін у вже існуючому коді.

3. Клієнтський додаток.

Створений з використанням Angular платформи. Завдяки раніше створеному Web Api, маємо можливість скористатися платформою Angular, та покласти відповідальність за користувальницький інтерфейс на інший проєкт. Структура клієнтського додатку виглядає як на рисунку 3.8.

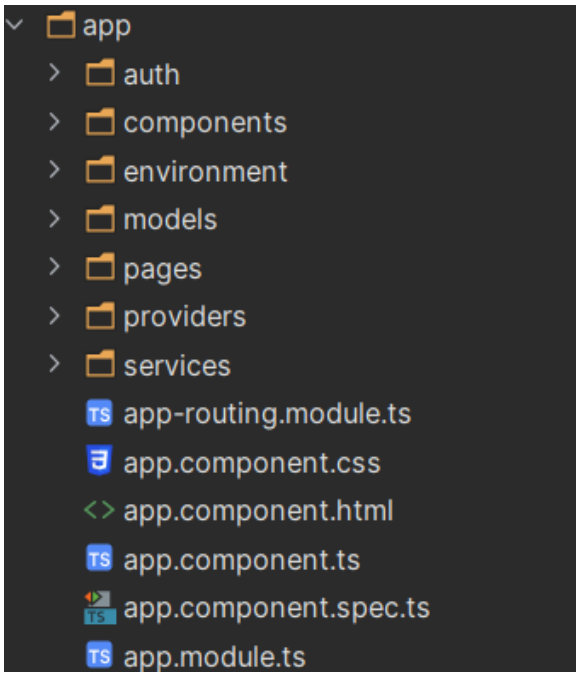


Рисунок 3.8 – Структура клієнтського додатку

Така структура дозволяє легко орієнтуватися в проєкті, додавати новий функціонал та редагувати вже існуючий. Цей проєкт використовує можливості двох раніше створених сервісів та об’єднує їх для представлення усіх функцій користувача.

3.2 Опис програмної реалізації. Інтерфейс користувача системи.

1. Інтерфейс сервісу авторизації.

З опису реалізації створено наступні сторінки:

1) Сторінка входу (рисунок 3.9).

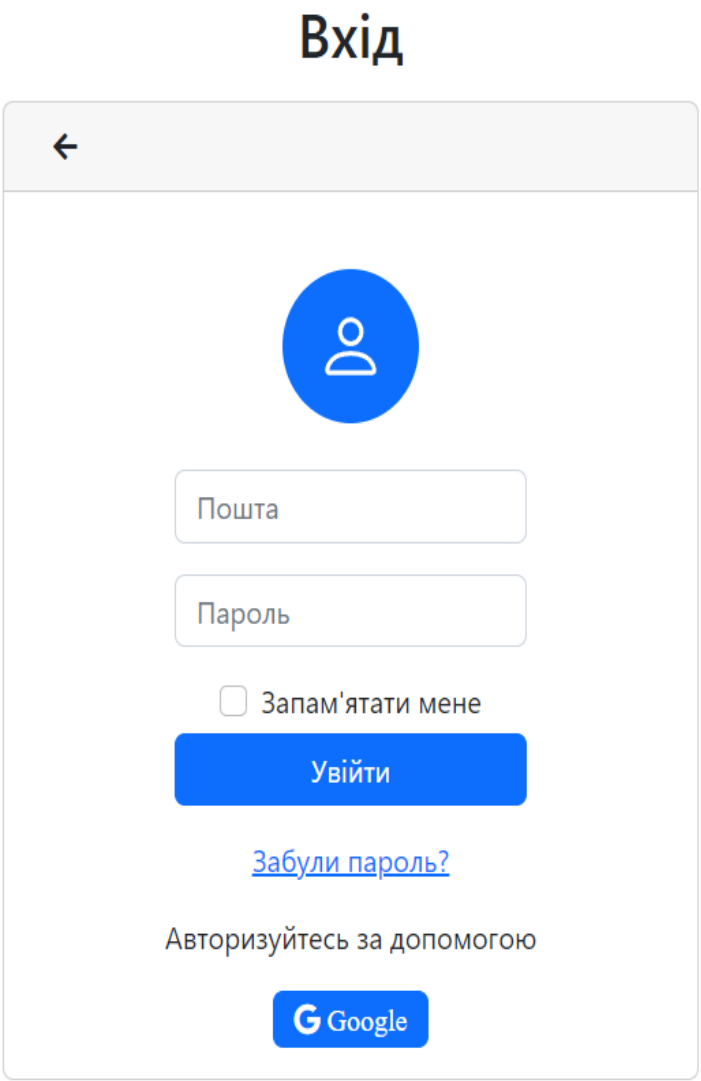


Рисунок 3.9 – Сторінка входу

2) Сторінка реєстрації (рисунок 3.10).

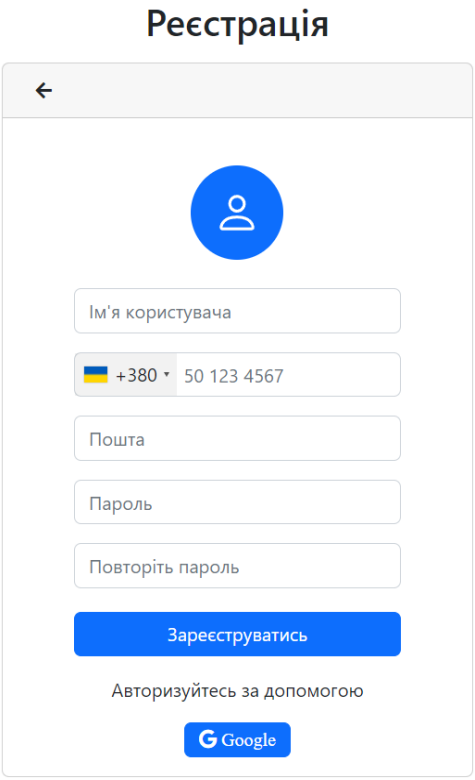


Рисунок 3.10 – Сторінка реєстрації

3) Сторінка відновлення паролю (рисунок 3.11).

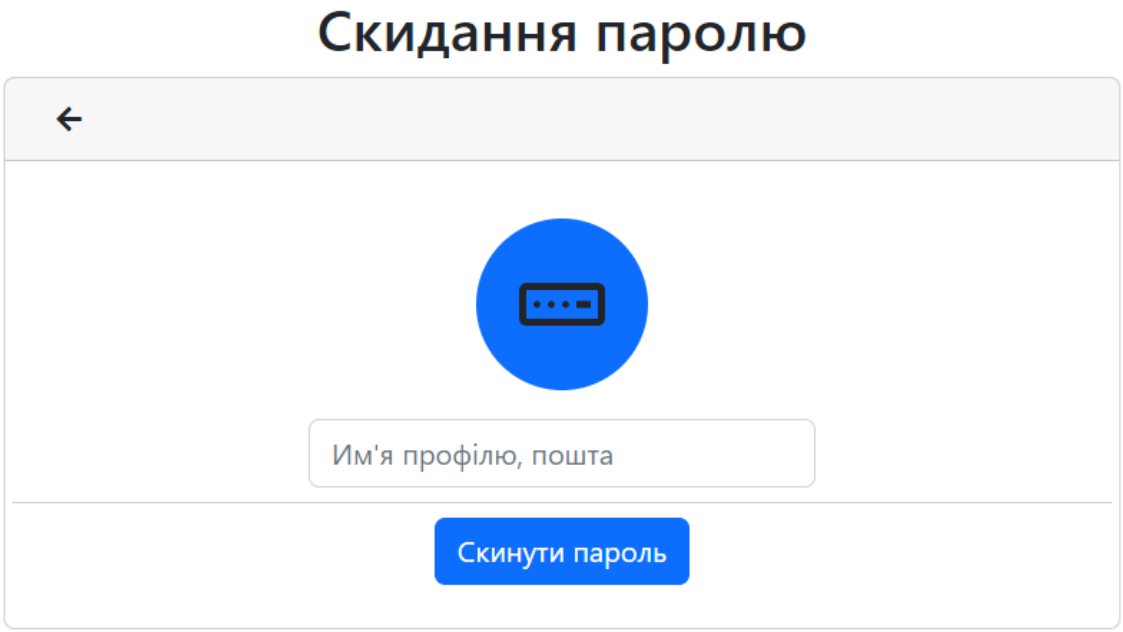


Рисунок 3.11 – Сторінка відновлення паролю

4) Сторінка профілю (рисунок 3.12).

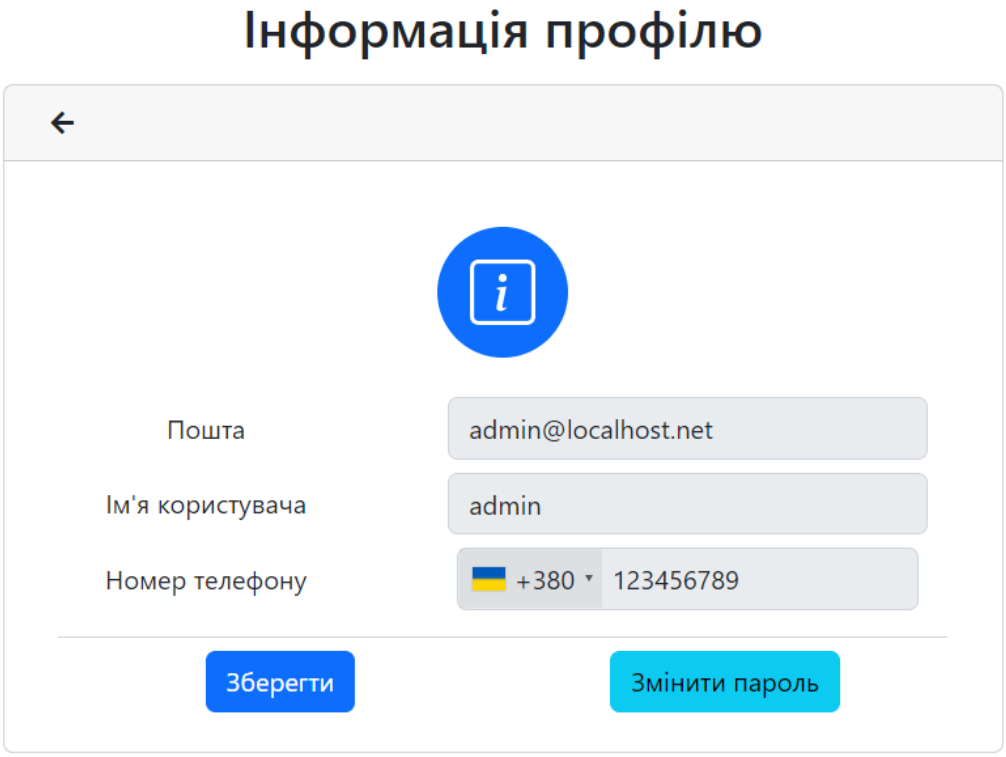


Рисунок 3.12 – Сторінка профілю

5) Сторінка змінення паролю (рисунок 3.13).

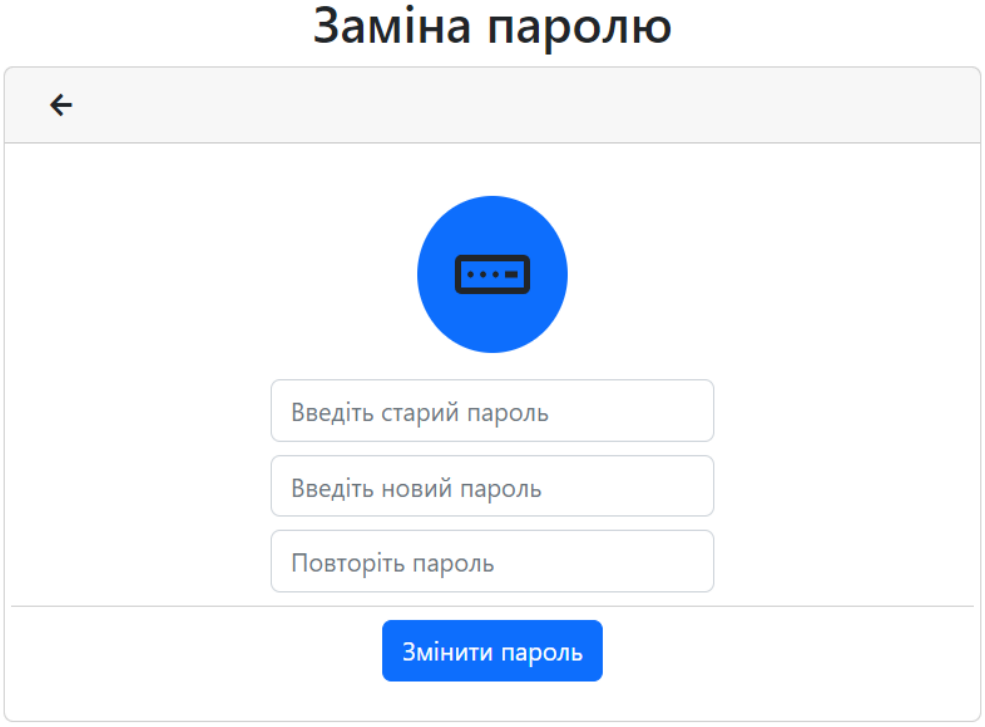


Рисунок 3.13 – Сторінка змінення паролю

2. Інтерфейс сервісу Web Api.

Сервіс використовує Web Api, отже звичного інтерфейсу він не має. На запити він повертає HTTP відповідь, що несе в собі, окрім іншого, код відповіді та тіло. Тіло, зазвичай, представляє собою JSON об’єкт. Приклад відповіді одного з запитів, доступного на сервері наведено на рисунку 3.14.

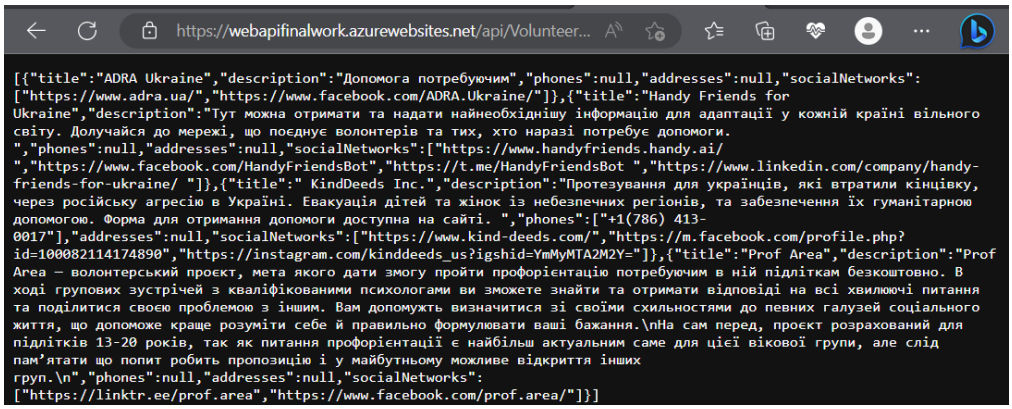


Рисунок 3.14 – Приклад запиту до Web Api

3. Інтерфейс клієнтського додатку.

Наведемо основні сторінки з клієнтського додатку (рисунок 3.14-3.18).

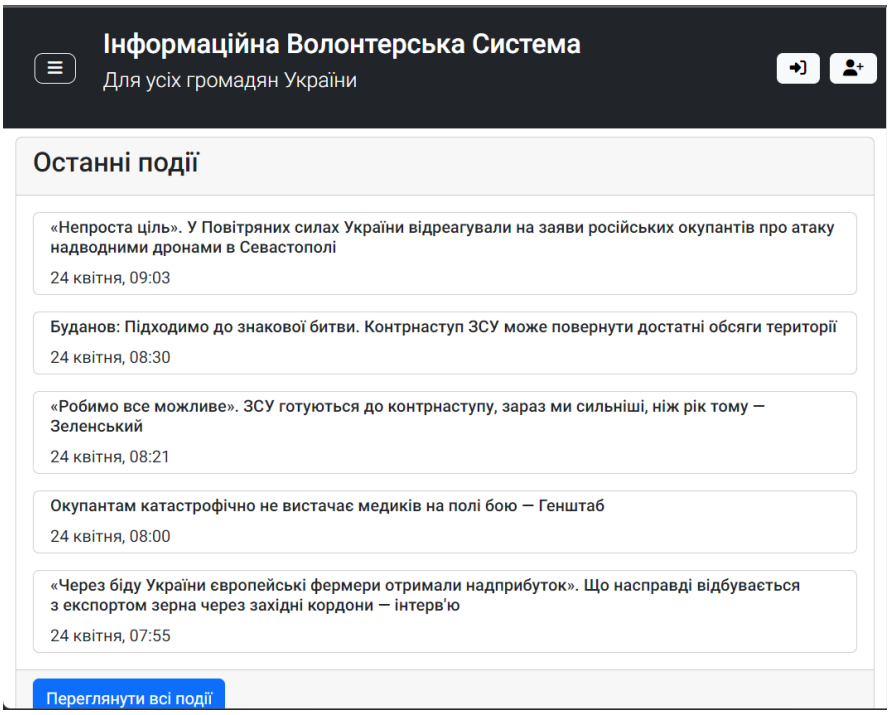


Рисунок 3.15 – Головна сторінка

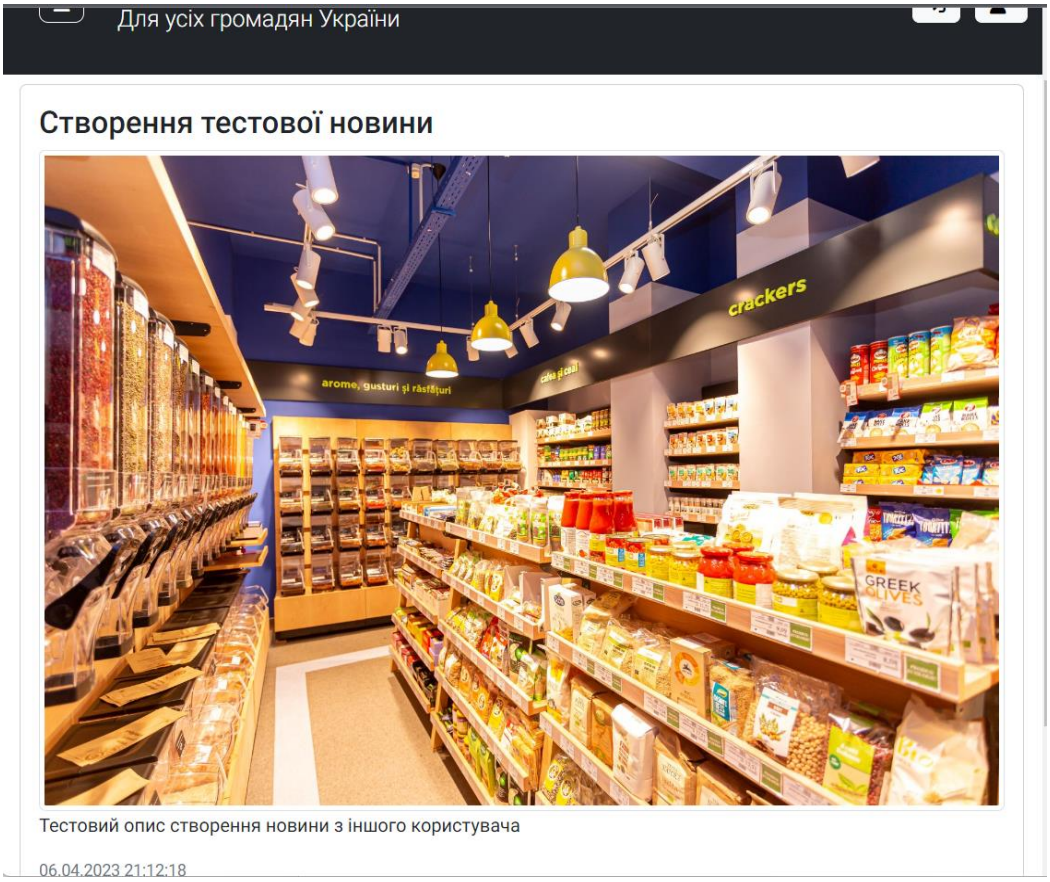


Рисунок 3.16 – Перегляд новини

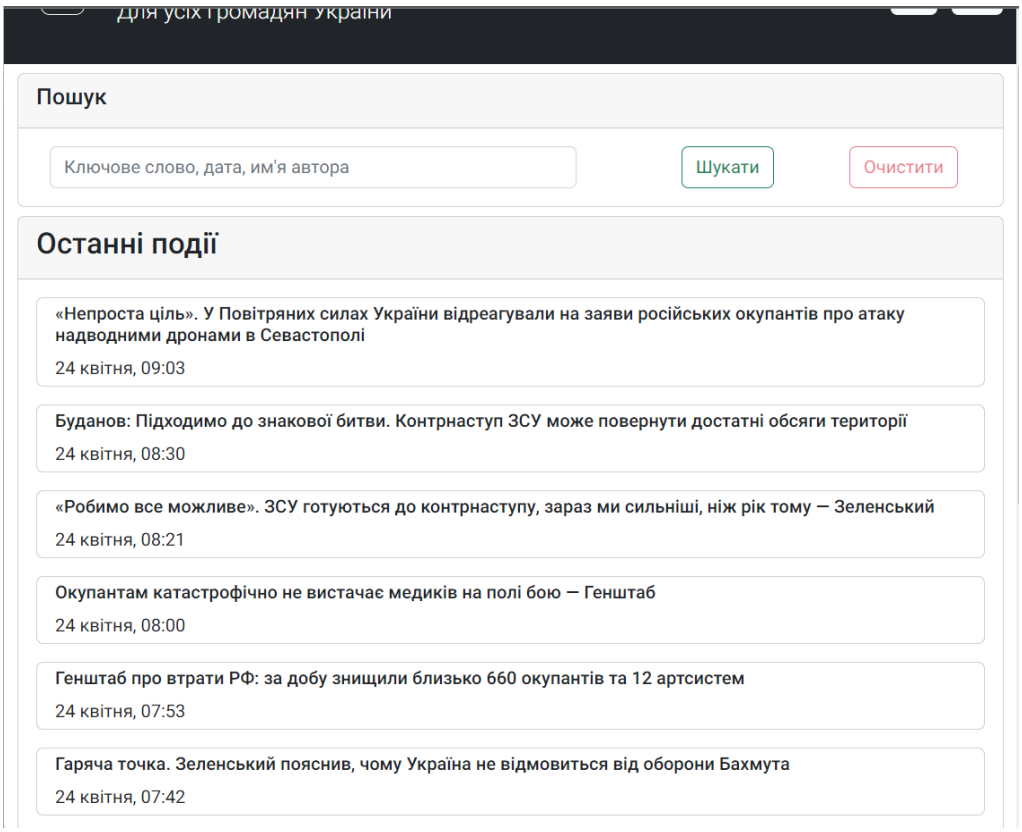


Рисунок 3.17 – Перегляд останніх подій

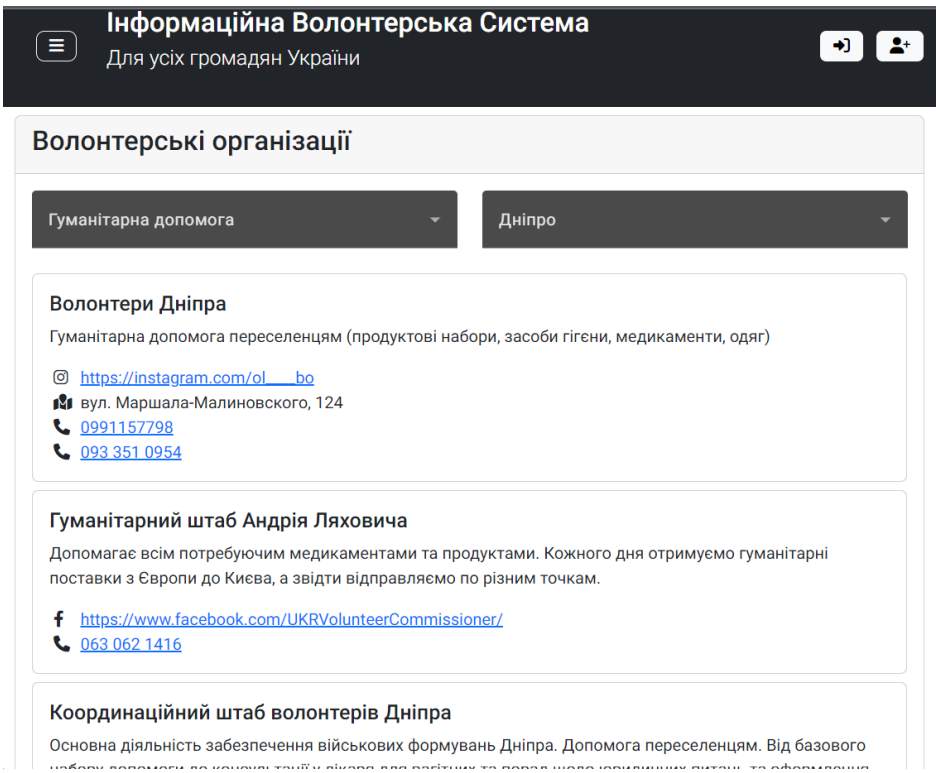


Рисунок 3.18 – Перегляд волонтерських організацій

Окрім цього, після авторизації користувачу доступні наступні сторінки і можливості (рисунок 3.19-3.20).

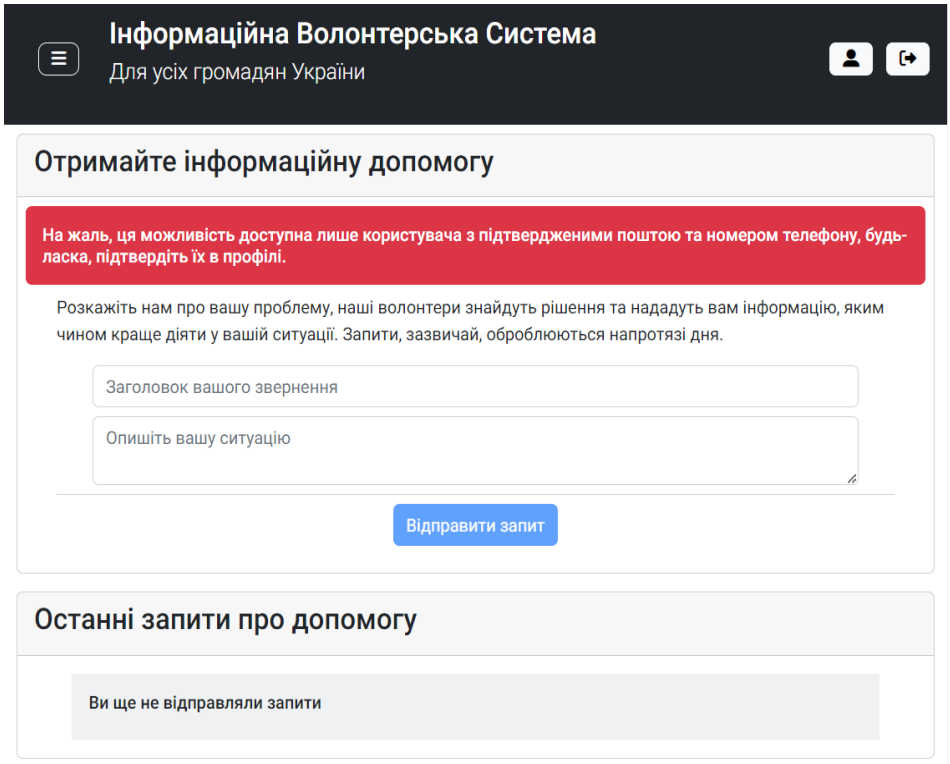


Рисунок 3.19 – Створення запиту про допомогу

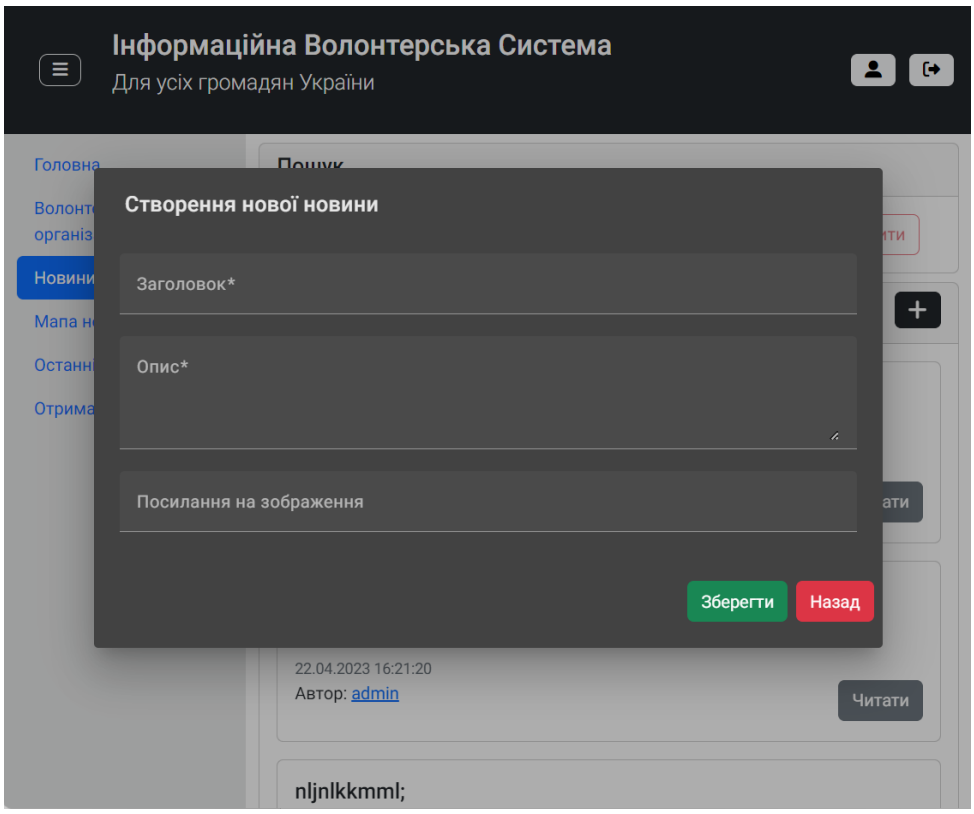


Рисунок 3.20 – Створення новини

Адміністратори і модератори мають додатковий функціонал (рисунок 3.21-3.22)

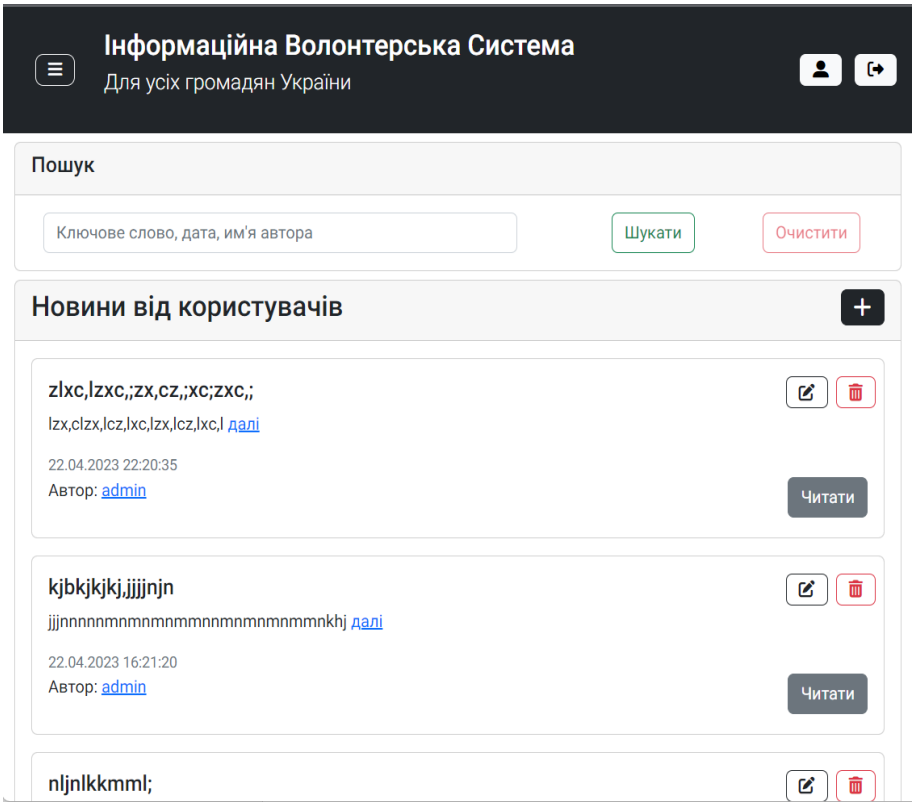


Рисунок 3.21 – Новини. Редагування, видалення.

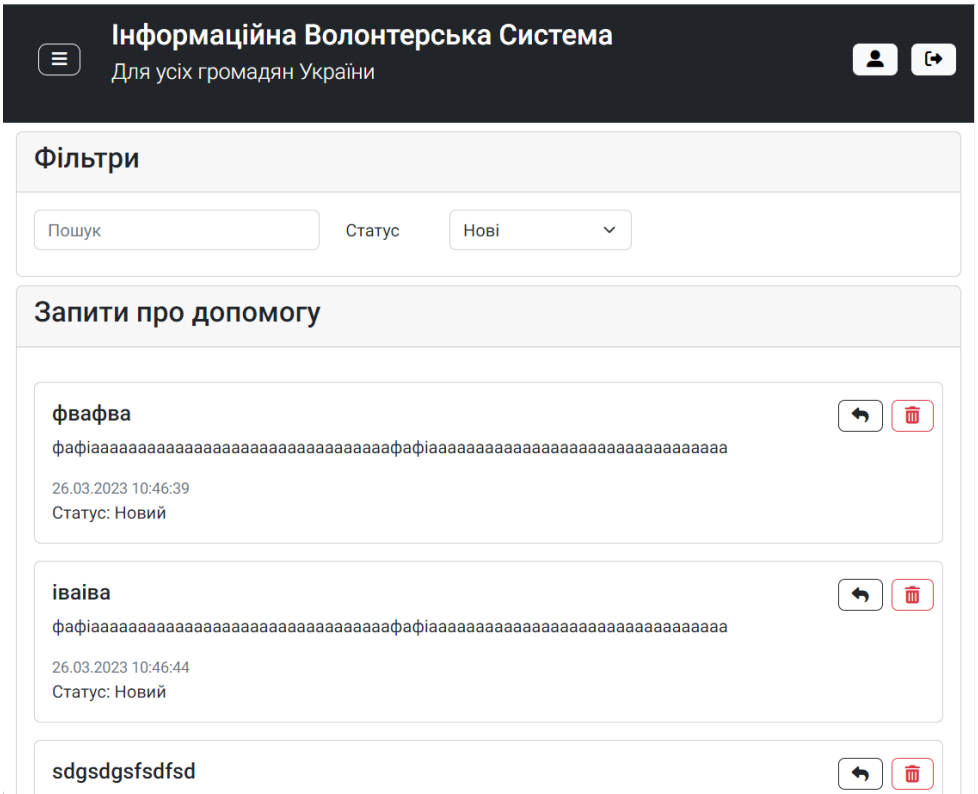


Рисунок 3.22 – Обробка запитів про допомогу

3.3 Методика роботи користувача з програмним продуктом

1. Системні вимоги

Оскільки сервіс доступний у Web, то користувачу необхідний доступ до інтернету та браузер, чого буде достатньо для роботи сервісу.

2. Сценарій роботи користувача з системою

Після переходу користувача за посиланням сервісу, він отримує доступ до перегляду новин, подій, отримання інформації щодо волонтерських організацій та мапи незламності – ці функції доступні для всіх користувачів.

У разі, якщо користувач потребує додаткової допомоги та хоче залишити запит про допомогу, йому необхідно зареєструватись, для чого він повинен натиснути на кнопку реєстрації (рисунок 3.23).

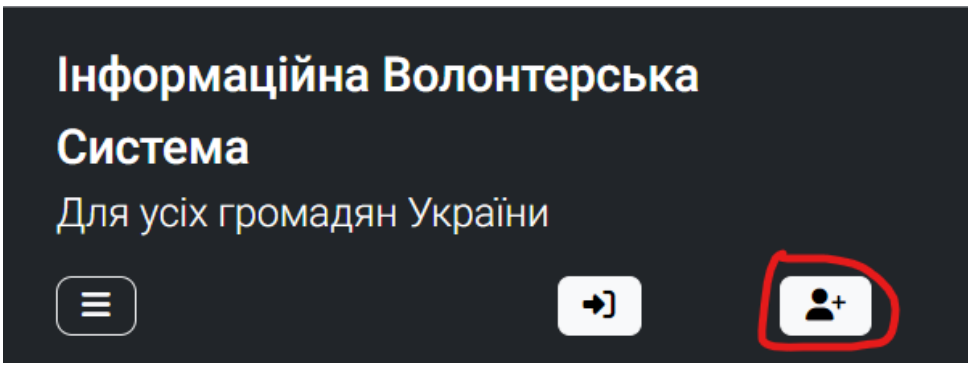


Рисунок 3.23 – Кнопка реєстрації

Вона є доступною на будь-якій сторінці сервісу до тих пір, поки користувач не авторизується.

Після натискання виклику цієї функції користувача буде перенаправлено на сервер авторизації для проходження реєстрації. При введенні невірних даних, користувач отримає повідомлення про це. Наприклад, при введені некоректного номеру телефону, користувач побачить повідомлення на рисунку 3.24.

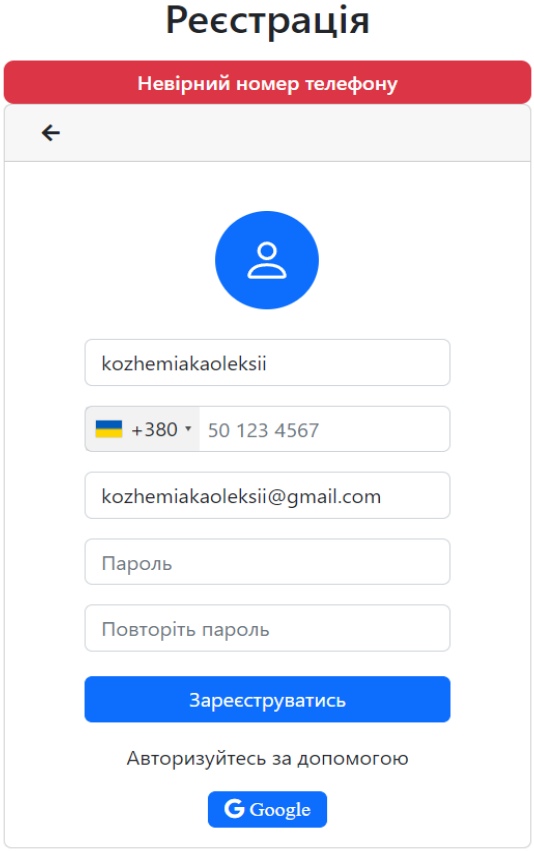


Рисунок 3.24 – Повідомлення про невірний номер телефону

У разі введення вірних усіх реєстраційних даних, то користувача перенаправить на сторінку входу для того, щоб він зміг увійти у свій новостворений профіль.

Після входу у новий профіль, користувача переправить на клієнтський додаток та відкриє функцію створення новин та залишення запиту про допомогу.

Для залишення запиту користувачу необхідно перейти у відповідний пункт меню. Однак спершу, він побачить повідомлення про помилку, через те, що має непідтверджені номер телефону та пошту (рисунок 3.25).

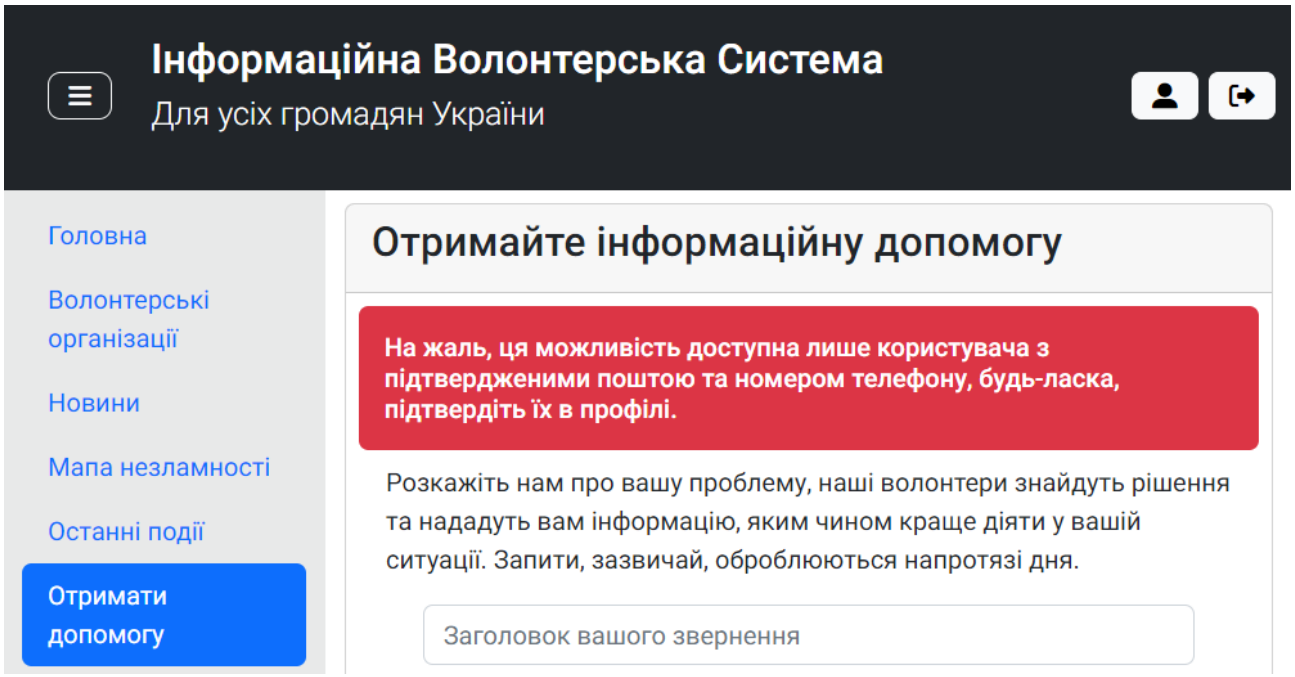


Рисунок 3.25 – Сторінка запиту про допомогу

Для підтвердження облікових даних користувачу необхідно перейди до свого профілю, посилання на яку доступне у тому місці, де раніше була кнопка входу в сервіс (рисунок 3.26).

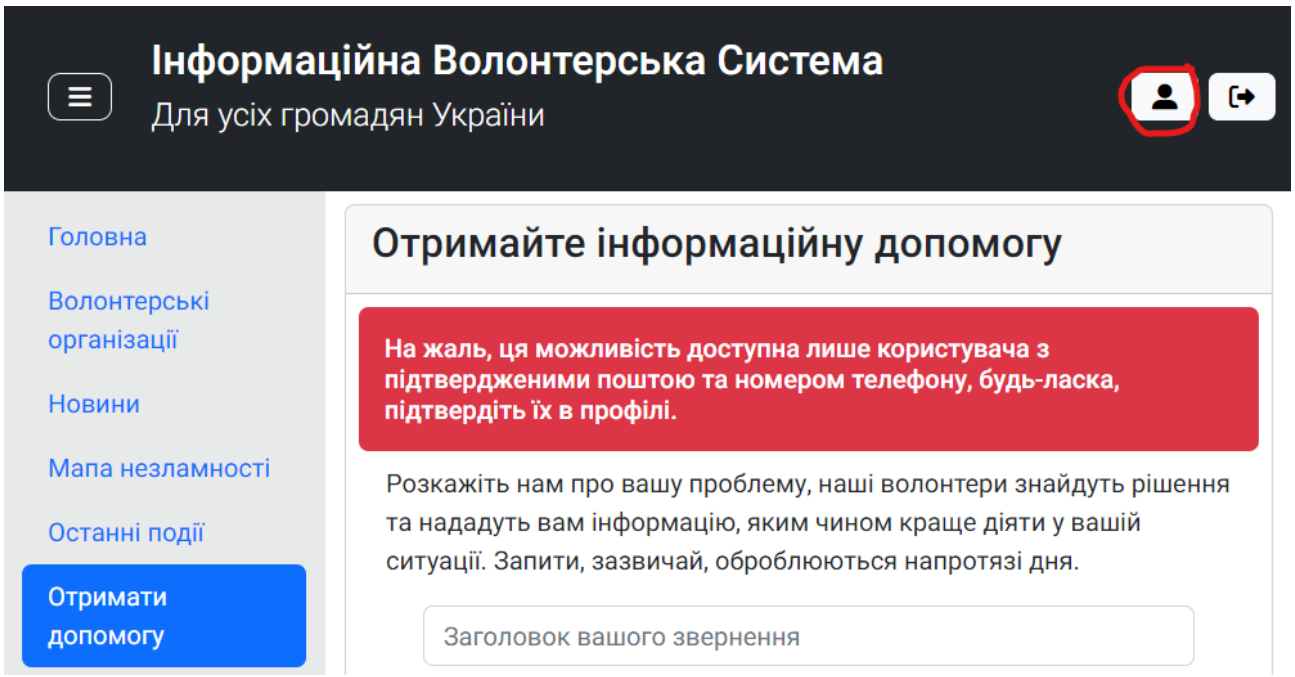


Рисунок 3.26 – Кнопка переходу до інформації профілю

На сторінці профілю він побачить інформацію, що було введено під час реєстрації, однак поряд з поштою на номером телефону буде відображатись інформація про те, що вони не підтверджені. Для підтвердження необхідно натиснути на посилання, що розміщенні під полями пошти та номеру телефону. (рисунок 3.27)

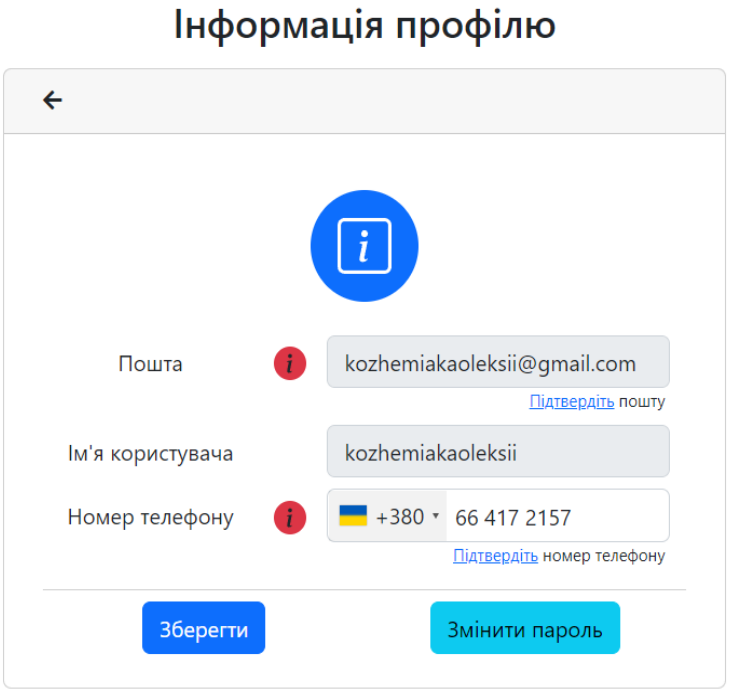


Рисунок 3.27 – Сторінка інформації профілю

Після натиснення підтвердження номеру телефону, користувача буде перекинуто на іншу сторінку, де буде доступне лише поле для вводу коду, що було вислане на телефон (рисунок 3.28).

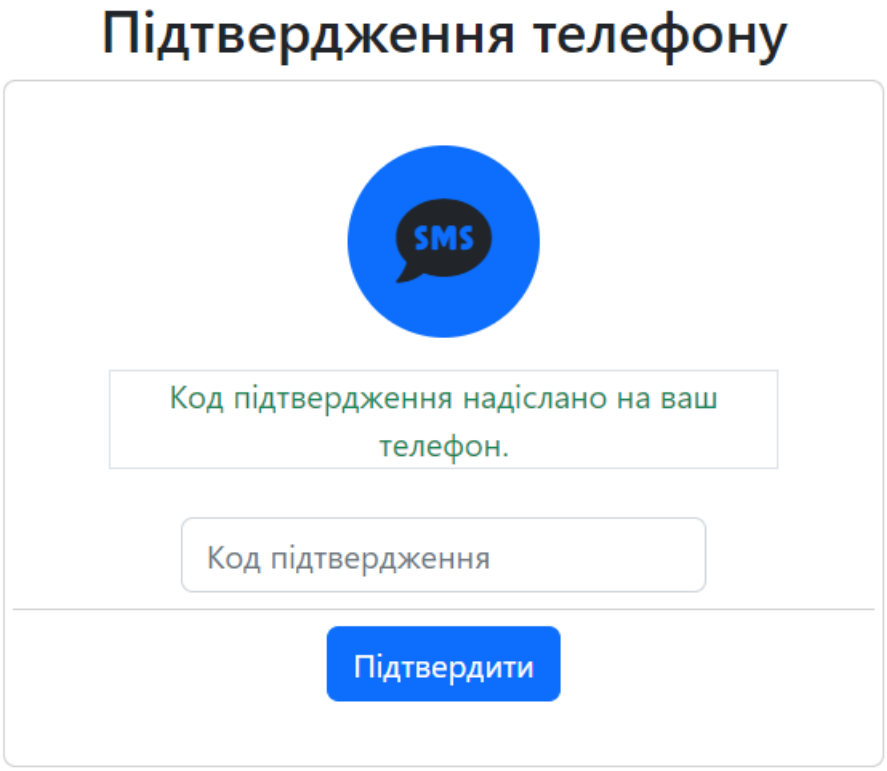


Рисунок 3.28 – Сторінка з введенням коду підтвердження

На мобільний телефон повинно надійти повідомлення (рисунок 3.29).

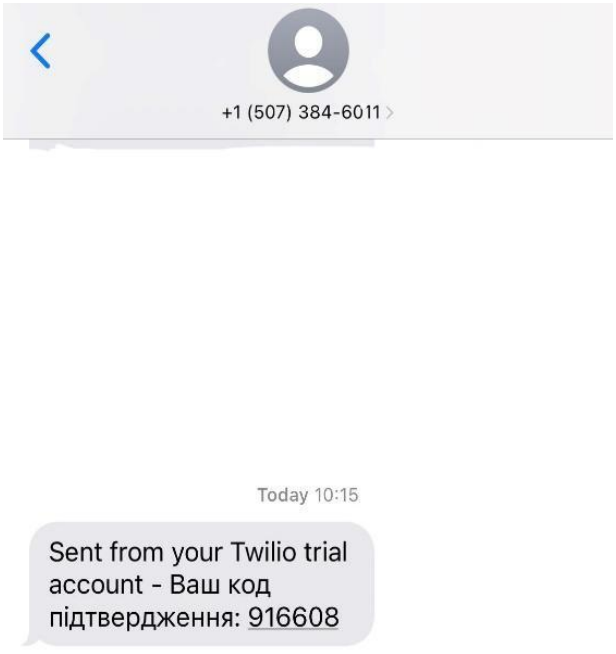


Рисунок 3.29 – Смс з кодом підтвердження

Після вводу отриманого коду, користувача буде перекинуто на сторінку профілю, де поле вводу номеру телефона стане неактивним і зникне повідомлення про необхідність підтвердження телефону (рисунок 3.30).

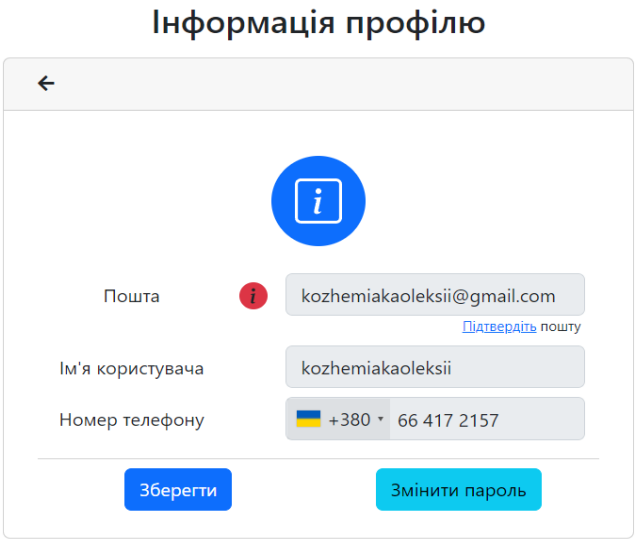
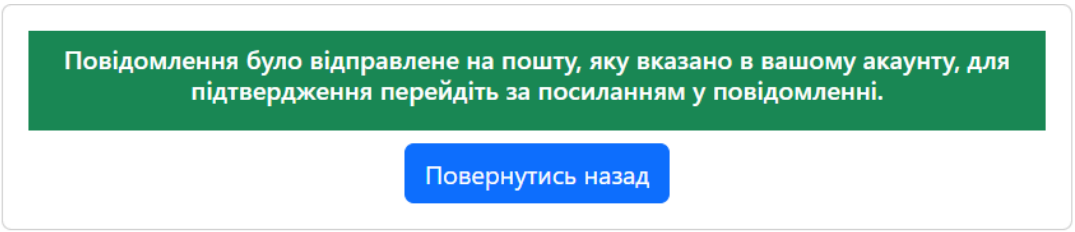


Рисунок 3.30 – Номер телефону підтверджено

Для підтвердження пошти, після натиснення посилання, користувач отримає повідомлення на рисунку 3.31.



Риунок 3.31 – Успішне надсилання повідомлення на пошту

На пошту він отримає повідомлення на рисунку 3.32.

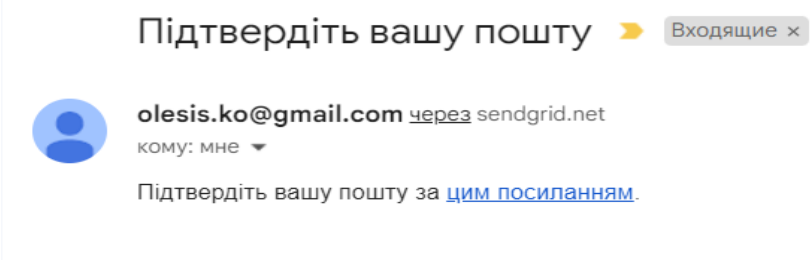


Рисунок 3.32 – Повідомлення підтвердження пошти

Після переходу за посиланням, користувача буде перенаправлено на профіль (рисунок 3.33).

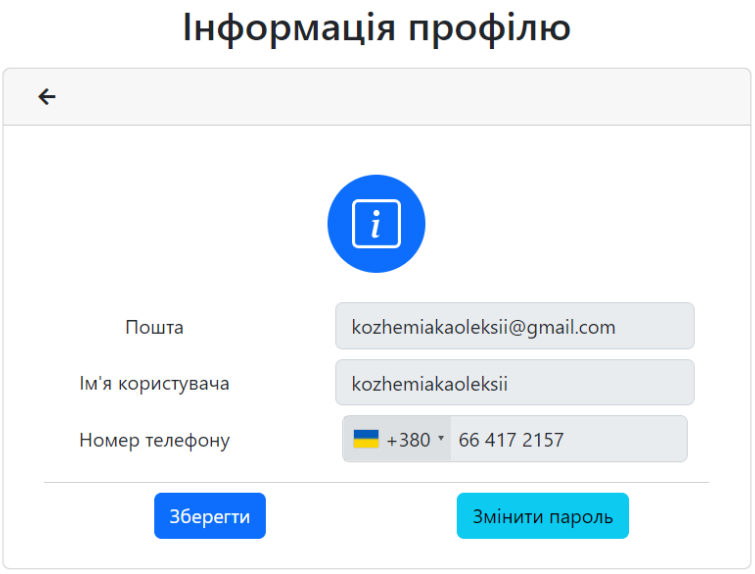


Рисунок 3.33 – Підтверджена інформація профілю

Тепер користувач зможе залишити запит про допомогу, для цього йому необхідно повернутись у основний додаток, та знову перейти до пункту меню отримати допомогу. Наразі, поле вводу доступне та повідомлення зникло (рисунок 3.34). У разі якщо повідомлення про не підтвердженність даних все ще відображається, необхідно виконати вихід та вхід в аккаунт, для отримання останніх актуальних даних с серверу автентифікації.

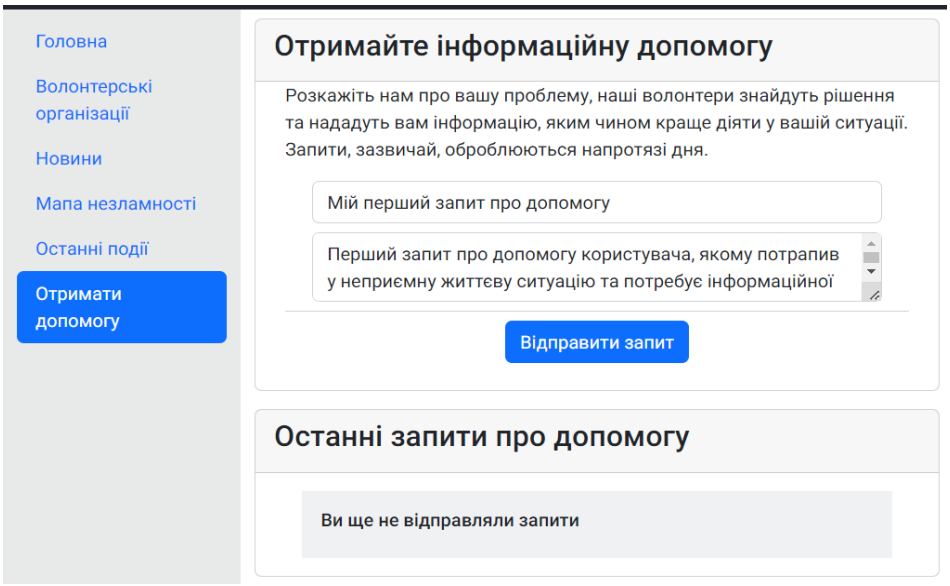


Рисунок 3.34 – Відправка запиту

Після натискання кнопки «Відправити запит», користувач побачить повідомлення про те, що його запит створено та отримає номер цього запиту. Номер він може використовувати у майбутньому для посилань в нових запитах (рисунок 3.35).

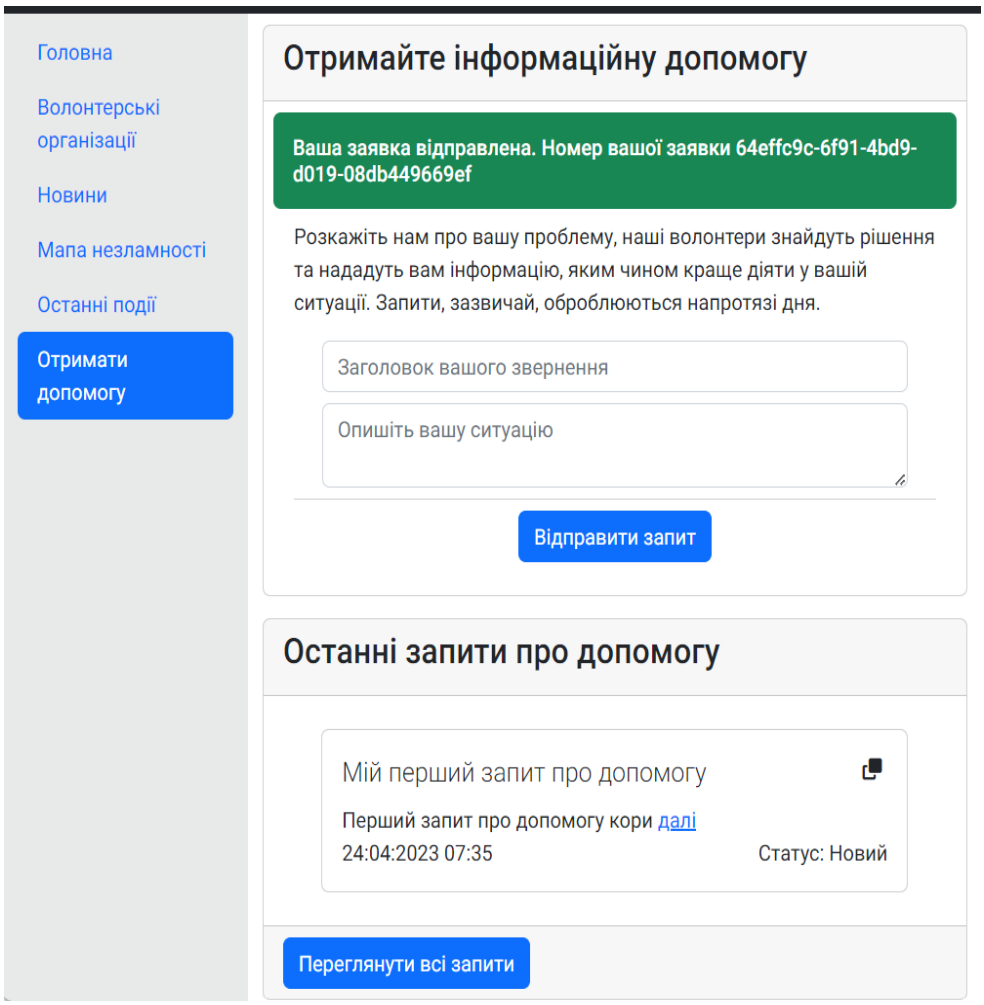


Рисунок 3.35 – Запит надіслано

Після надіслання запиту його буде оброблено у порядку черги адміністраторами сервісу, відповідь надійде на пошту, що вказана у профілі, у відповіді будуть вказані відомості відповідно до запиту користувача, що зможуть допомогти йому.

Користувач зможе відслідковувати статус запиту на сторінці (рисунок 3.35), після опрацювання та надіслення відповіді статус запиту зміниться на «оброблений».

3.4 Висновки до третього розділу

За результатами розділу створено програмний продукт у вигляді веб-сервісу з відкритим вихідним кодом, використовуючи платформи ASP.NET та Angular. Виконуючи завдання дипломної роботи розроблено функціонал, що надає користувачам доступ до інформації про волонтерські пункти, останні новини, можливість отримати персональну інформаційну допомогу. Окремо, створено сервіс автентифікації, що виконує управління профілями користувачів та в майбутньому дозволить створювати різні сервіси під управлінням одного профілю.

Таким чином, розроблений веб-сервіс повністю відповідає постановці завдання, реалізує її в повному обсязі, при його розробці використовувались актуальні архітектурні підходи до розробки програмного забезпечення, які дозволять у майбутньому легко підтримувати та розвивати систему.

ВИСНОВКИ

За результатами роботи проведено аналіз предметної області, в процесі аналізу було сформульовано постановку проблеми, завдання. Аналізуючи існуючі реалізаціях, було визначено основні недоліки цих систем та розроблено вимоги, які повинен задовольнити веб-сервіс.

В процесі аналізу сучасних методів та засобів розробки визначений перелік технологій, що необхідні для вирішення завдання, обґрунтована мета їх використання та призначення. Обрано мінімальний перелік необхідних засобів для досягнення мети роботи.

Під час проєктування архітектури оглянуті архітектурні підходи до розробки веб-сервісів, обрані найбільш вдалі, що дозволили у короткі терміни створити високонавантажені відмовостійкі сервіси. Проаналізовано використання MVC та Clean architecture у створенні сучасних високонавантажених веб-сервісів.

Розроблено веб-сервіс інформаційної підтримки цивільного населення з використанням технологій ASP.NET та Angular та базуючись на архітектурі Clean architecture та MVC.

При реалізації завдання використовувались додаткові сервіси для SMS-повідомлень, email-повідомлень, валідації номерів телефону. Кожна окремо ці операції є досить складними та потребують додаткової інтеграції з мобільними операторами, email-сервісами, тому використання сторонніх сервісів перекладає відповідальність за реалізацію цих можливостей на них та дозволяє сфокусувати увагу розробників на важливіших речах.

Як сховище даних використовується MsSQL Server, який забезпечить безвідмовну роботу при великих навантаженнях, зручний доступ до бази даних, а використання хмарного сервісу від Azure дозволить легко адмініструвати сервіс, збільшуючи чи зменшуючи необхідні розрахункові можливості за необхідністю.

Під час проєктування бази даних був використаний підхід Code first, що дозволяє створювати моделі у вигляді C# класів, які пізніше за допомогою можливостей Entity Framework Tools будуть перетворені на відповідні SQL запити задля створення цих моделей. Використовуючи такий підхід, розробники мають змогу зосередитись на створенні бізнес логіки та відсторонитись від написання безпосередньо SQL запитів, що в свою чергу пришвидшує та полегшує розробку.

Проєкт створено з відкритим вихідним кодом у репозиторії GitHub.

Розміщення проєкт у публічному репозиторії дозволить залучити інших розробників до покращення сервісу, знаходження неточностей, виправлення. Виходячи з волонтерської ідеї проєкт, ці фактори повинні забезпечити бажання небайдужих розробників підтримати подальше існування системи, її розвиток та підтримку. Шляхами цього розвитку може бути:

1. Розширення функціоналу зареєстрованих користувачів – створення додаткових ролей, як приклад, «модератор», покращення доступу до ресурсів базованого на ролях. Оскільки наразі існує лише 2 ролі адміністратор та користувача, то і поділ доступу розглядається лише у такому варіанті.

2. Створення мобільного додатку з доступом в офлайні. Він повинен стати базою знань у телефоні користувача. Доступ в офлайні повинен буде реалізований, як приклад, наступним чином, при доступі до інтернету користувач отримує останню інформацію, наприклад, за місяць, вона зберігається автоматично на телефоні, з інтернетом користування додатком не повинно суттєво відрізнятися від користування основного сервісу. У разі відсутності доступу до веб-серверу, додаток переходить у автономний режим та використовує збережену на телефоні користувача інформації у якості бази даних, замість веб-серверу. Функціонал повинен бути максимально доступним, користувач повинен мати змогу виконувати усі дії, які міг з інтернетом, після того, як зв’язок користувача с сервером налагоджено – повинна відбутися синхронізація, в процесі якої оновлені дані з телефону користувача повинні надійти до серверу, а сервер надасть свої дані.

3. Інтеграція автентифікації з Дією. Дія набула достатньої популярності та забезпечить авторизацію лише унікальних користувачів, а для звичайних громадян пришвидшить виконання реєстраційних дій.

надання доступу до ресурсів проєкту суспільству, створенні вичерпної інструкції для користувачів.

Проєкт було розгорнуто у хмарних сервісах Azure, які забезпечать цілодобовий доступ до нього, покращують процес розгортання нових версій сервісу завдяки тісній інтеграції з засобами розробки, створять можливості для його масштабування.

Для збільшення безпечності використання сервісу усі ключі та паролі, що використовуються сервісу, як наприклад, пароль від користувача бази даних, зберігаються у сервісі Key Vault Azure, який забезпечує безпеку використання чутливих даних завдяки максимальному обмеженню доступу до них.

Мета та завдання кваліфікаційної роботи були виконані у повному обсязі. В результаті отриманий веб-сервіс забезпечує цілодобовий доступ до актуальної інформації та дозволяє отримати інформаційну допомогу.

Економічною складовою цього проєкту може стати створення фонду проєкту, в який небайдужі громадяни зможуть надсилати кошти на його підтримку та покриття витрат на сервери, домени та інше.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Welcome to C# 11 – .NET Blog. URL: https://devblogs. microsoft.com/dotnet/welcome-to-csharp-11.

2. C# 9 and .NET 5 – Modern Cross-Platform Development, Fifth Edition by Mark J. Price. URL: https://books.google.com.ua/books/about /C_9_and_NET_5_Modern_Cross_Platform_Deve.html?id=00EIEAAAQBAJ&redir_esc=y.

3. What is ASP.NET Core? | .NET. URL: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core.

4. Rider: The Cross-Platform .NET IDE from JetBrains. URL: https://www.jetbrains.com/rider.

5. What Are CSS Vendor or Browser Prefixes? URL: https://www.thoughtco.com/css-vendor-prefixes-3466867.

6. Bootstrap · The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com.

7. Announcing TypeScript 4.0 – TypeScript. URL: https://devblogs.microsoft.com/typescript/announcing-typescript-4-0.

8. What’s the difference between AngularJS and Angular? | Gorrion's Blog. URL: https://www.gorrion.io/blog/angularjs-vs-angular.

9. Angular v13 is now Available. We’re back with the brand new release… | Angular Blog. URL: https://blog.angular.io/angular-v13-is-now-available-cce66f7bc296.

10. Learning SQL: Generate, Manipulate, and Retrieve Data 3rd, Alan Beaulieu. URL: https://books.google.com.ua/books/about/Learning_SQL.html?id=Ml3UDwAAQBAJ&redir_esc=y.

11. Microsoft Docs – Entity Framework. URL: https://learn.microsoft.com/en-us/aspnet/entity-framework.

12. Clean Code, Robert Martin. URL: https://github.com/dev-marko/clean-code-book.

13. Refactoring, Martin Flower, Kent Back. Видавництво: Addison-Wesley Professional; 2nd edition.

14. Konrad Kokosa Pro .NET Memory Management, 528 с.

15. Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides Design Patterns: Elements of Reusable Object-Oriented Software, 395 с.

16. Library of Congress Cataloging-in-Publication Data Design Patterns : elements of reusable object-oriented software / Erich Gamma – Addison-Wesley professional computing series.

17. Офіційна документація ASP.NET. URL: https://dotnet.microsoft.com/learn/aspnet.

18. Офіційний сайт C#. URL: https://docs.microsoft.com/en-us/dotnet/csharp.

19. Офіційна документація Angular. URL: https://angular.io/docs.

20. Офіційна документація JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide.

21. Stack Overflow – популярний форум для програмістів. URL: https://stackoverflow.com.

22. ASP.NET MVC Tutorial – серія практичних руководств. URL: https://www.tutorialspoint.com/asp.net_mvc/index.htm.

23. Learn C# – навчальні матеріали по C#. URL: https://www.learn-c.org.

24. Bootstrap 4 Tutorial – навчальний курс з Bootstrap. URL: https://www.w3schools.com/bootstrap4.

25. Angular Tutorial – навчальний курс по Angular. URL: https://angular.io/tutorial.

26. MDN Web Docs – документація по веб-технологіям. URL: https://developer.mozilla.org.

27. C# Station – ресурс для навчання C#. URL: https://csharp-station.com.

28. Bootstrap Expo – галерея проєктів на Bootstrap. URL: https://expo.getbootstrap.com.

29. Angular University – онлайн-курси та ресурси по Angular. URL: https://angular-university.io.

30. JavaScript.info – навчальний матеріал по JavaScript. URL: https://javascript.info.

31. FreeCodeCamp – відеоуроки та навчальні матеріали з програмування. URL: https://www.freecodecamp.org.

32. Telerik UI for ASP.NET MVC – бібліотека компонентів для ASP.NET. URL: https://www.telerik.com/aspnet-mvc.

33. C# Corner – спільнота програмістів на C#. URL: https://www.c-sharpcorner.com.

34. BootstrapBay – шаблони та ресурси для Bootstrap. URL: https://bootstrapbay.com.

35. Angular Style Guide – рекомендації щодо стилю програмування в Angular. URL: https://angular.io/guide/styleguide.

36. Eloquent JavaScript – книга та онлайн-посібник з JavaScript. URL: https://eloquentjavascript.net.

37. Codecademy – інтерактивні курси програмування. URL: https://www.codecademy.com.

38. ASP.NET Core – офіційний сайт ASP.NET Core. URL: https://dotnet.microsoft.com/apps/aspnet.

39. C# Programming Yellow Book – безкоштовна книга з програмування на C#. URL: http://www.csharpcourse.com.

40. Bootstrap Cheat Sheet – швидкий посібник по використанню Bootstrap. URL: https://hackerthemes.com/bootstrap-cheatsheet.

41. Angular Architecture – курс про архітектуру Angular. URL: https://www.udemy.com/course/angular-architecture.

42. JavaScript Design Patterns – книга про патерни проєктування на JavaScript. URL: https://addyosmani.com/resources/essentialjsdesignpatterns/book.

43. Pluralsight – онлайн-платформа для навчання програмуванню та розробці. URL: https://www.pluralsight.com.

44. Microsoft Learn – навчальні матеріали та курси від Microsoft. URL: https://docs.microsoft.com/en-us/learn.

ДОДАТОК А

https://github.com/sakozoko/finalpaper/

ДОДАТОК Б

HelpRequestController.cs

```
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using WebApi.Features;
using WebApi.InputModels.HelpRequest;
using WebApiApplication.Features.HelpRequestFeatures.Commands;
using WebApiApplication.Features.HelpRequestFeatures.Queries;

namespace WebApi.Controllers;

[Authorize]
[Route("api/helprequest")]
public class HelpRequestController : ControllerBase
{
    private readonly IMediator _mediatr;

    public HelpRequestController(IMediator mediatr)
    {
        _mediatr = mediatr;
    }

    [HttpPost("")]
    public async Task<IActionResult> Create([FromBody] CreateHelpRequestInputModel model)
    {
        var command = new CreateHelpRequestCommand
        {
            Title = model.Title,
            Description = model.Description,
            UserId = User.Claims.GetGuidUserId(),
            EmailConfirmed = User.Claims.GetEmailConfirmed(),
            Username = User.Claims.GetUserName(),
            UserEmail = User.Claims.GetUserEmail()
        };
        var result = await _mediatr.Send(command);
        return Ok(result);
    }

    [HttpGet("")]
    public async Task<IActionResult> Get([FromQuery] int page, [FromQuery] int pageSize)
    {
        var query = new GetHelpRequestForUserByPageQuery
        {
            UserId = User.Claims.GetGuidUserId(),
            Page = page,
            PageSize = pageSize
        };
        var result = await _mediatr.Send(query);
        return Ok(result);
    }
```

```
    [HttpGet("count")]
    public async Task<IActionResult> GetCountForUser()
    {
        var query = new GetHelpRequestCountForUserQuery
        {
            UserId = User.Claims.GetGuidUserId()
        };
        var result = await _mediatr.Send(query);
        return Ok(result);
    }

    [HttpGet("search")]
    public async Task<IActionResult> SearchForUser([FromQuery] string filter, [FromQuery] int
page,
        [FromQuery] int pageSize)
    {
        var query = new GetHelpRequestsByFilterQuery
        {
            UserId = User.Claims.GetGuidUserId(),
            Filter = filter,
            Page = page,
            PageSize = pageSize
        };
        var result = await _mediatr.Send(query);
        return Ok(result);
    }

    [HttpGet("search/count")]
    public async Task<IActionResult> SearchCount([FromQuery] string filter)
    {
        var query = new GetHelpRequestCountByFilterQuery
        {
            UserId = User.Claims.GetGuidUserId(),
            Filter = filter
        };
        var result = await _mediatr.Send(query);
        return Ok(result);
    }

    [Authorize("Admin")]
    [HttpGet("getall")]
    public async Task<IActionResult> GetAll(GetHelpRequestsQuery query)
    {
        var result = await _mediatr.Send(query);
        return Ok(result);
    }

    [Authorize("Admin")]
    [HttpGet("getall/count")]
    public async Task<IActionResult> GetAllCount(GetHelpRequestCountQuery query)
    {
```

```
        var result = await _mediatr.Send(query);
        return Ok(result);
    }

    [Authorize("Admin")]
    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete(DeleteHelpRequestCommand command)
    {
        var result = await _mediatr.Send(command);
        return Ok(result);
    }

    [Authorize("Admin")]
    [HttpPut("answer")]
    public async Task<IActionResult> Answer([FromBody] AnswerToHelpRequestCommand
command)
    {
        var result = await _mediatr.Send(command);
        return Ok(result);
    }
}
```

LatestNewsController.cs

```
using MediatR;
using Microsoft.AspNetCore.Mvc;
using WebApiApplication.Features.LatestNewFeatures.Queries;

namespace WebApi.Controllers;

[Route("api")]
public class LatestNewsController : ControllerBase
{
    private readonly IMediator _mediator;

    public LatestNewsController(IMediator mediator)
    {
        _mediator = mediator;
    }

    [HttpGet("latestnews")]
    public async Task<IActionResult> GetLatestNews(GetLatestNewsByPageQuery command)
    {
        return Ok(await _mediator.Send(command));
    }

    [HttpGet("latestnews/count")]
    public async Task<IActionResult> GetLatestNewsCount(GetLatestNewsCountQuery
command)
    {
        return Ok(await _mediator.Send(command));
    }
```

```
    [HttpGet("latestnews/filter")]
    public async Task<IActionResult> GetLatestNewsByFilter(GetLatestNewsByFilterQuery
command)
    {
        return Ok(await _mediator.Send(command));
    }
}
```

PublicNewController.cs

```
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using WebApi.Features;
using WebApi.InputModels.PublicNew;
using WebApiApplication.Features.PublicNewFeatures.Commands;
using WebApiApplication.Features.PublicNewFeatures.Queries;

namespace WebApi.Controllers;

[Route("api/publicnew")]
public class PublicNewController : ControllerBase
{
    private readonly IMediator _mediator;

    public PublicNewController(IMediator mediator)
    {
        _mediator = mediator;
    }

    [HttpGet("")]
    public async Task<IActionResult> Get([FromQuery] GetPublicNewsByPageQuery query)
    {
        var result = await _mediator.Send(query);
        return Ok(result);
    }

    [HttpGet("count")]
    public async Task<IActionResult> GetCount()
    {
        var query = new GetPublicNewsCountQuery();
        var result = await _mediator.Send(query);
        return Ok(result);
    }

    [HttpGet("search")]
    public async Task<IActionResult> Search([FromQuery] GetPublicNewsByFilterQuery query)
    {
        var result = await _mediator.Send(query);
        return Ok(result);
    }

    [HttpGet("search/count")]
```

```
    public async Task<IActionResult> SearchCount([FromQuery] GetPublicNewsCountByFilterQuery query)
    {
        var result = await _mediator.Send(query);
        return Ok(result);
    }

    [HttpGet("{id}")]
    public async Task<IActionResult> GetById([FromRoute] Guid id)
    {
        var query = new GetPublicNewByIdQuery
        {
            Id = id
        };
        var result = await _mediator.Send(query);
        return Ok(result);
    }

    [Authorize]
    [HttpPost("")]
    public async Task<IActionResult> Create([FromBody] CreatePublicNewInputModel inputModel)
    {
        var username = User.Claims.GetUserName();
        var userId = User.Claims.GetGuidUserId();
        if (username == null || userId == null) return BadRequest("User not found");
        var command = new CreatePublicNewCommand
        {
            Title = inputModel.Title,
            Description = inputModel.Description,
            ImageUrl = inputModel.ImageUrl,
            UserId = userId.Value,
            Author = username,
            CreatedAt = inputModel.CreatedAt ?? DateTime.UtcNow
        };
        var result = await _mediator.Send(command);
        return Ok(result);
    }

    [Authorize("Admin")]
    [HttpPut("")]
    public async Task<IActionResult> Update([FromBody] UpdatePublicNewInputModel inputModel)
    {
        var command = new UpdatePublicNewCommand()
        {
            Title = inputModel.Title,
            Description = inputModel.Description,
            ImageUrl = inputModel.ImageUrl,
            Id = inputModel.Id,
            CreatedAt = inputModel.CreatedAt,
            Author = inputModel.Author
```

```
        };
        var result = await _mediator.Send(command);
        return Ok(result);
    }

    [Authorize("Admin")]
    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete([FromRoute] Guid id)
    {
        var command = new DeletePublicNewCommand
        {
            Id = id
        };
        var result = await _mediator.Send(command);
        return Ok(result);
    }
}
```

VolunteerOrganizationController.cs

```
using MediatR;
using Microsoft.AspNetCore.Mvc;
using WebApiApplication.Features.VolunteerOrganizationFeatures.Queries;

namespace WebApi.Controllers;

[ApiController]
[Route("api/[controller]")]
public class VolunteerOrganizationController : ControllerBase
{
    private readonly IMediator _mediator;

    public VolunteerOrganizationController(IMediator mediator)
    {
        _mediator = mediator;
    }

    [HttpGet]
    public async Task<IActionResult> GetVolunteerOrganizations([FromQuery]
GetVolunteerOrganizationsQuery command)
    {
        return Ok(await _mediator.Send(command));
    }

    [HttpGet("availableCities")]
    public async Task<IActionResult> GetAvailableCities([FromQuery] GetAvailableCitiesQuery
command)
    {
        return Ok(await _mediator.Send(command));
    }

    [HttpGet("availableCategories")]
```

```
    public async Task<IActionResult> GetAvailableCategories([FromQuery]
GetAvailableCategoriesQuery command)
    {
        return Ok(await _mediator.Send(command));
    }

    [HttpGet("count")]
    public async Task<IActionResult> GetCount([FromQuery]
GetVolunteerOrganizationsCountQuery command)
    {
        return Ok(await _mediator.Send(command));
    }
}
```

AnswerToHelpRequestCommand.cs

```
using AutoMapper;
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using WebApiApplication.Context;
using WebApiApplication.Features.HelpRequestFeatures.Dto;
using WebApiApplication.Shared.Exceptions;
using WebApiCore.Models;

namespace WebApiApplication.Features.HelpRequestFeatures.Commands
{
    public class AnswerToHelpRequestCommand : IRequest<HelpRequestDto>
    {
        public Guid Id { get; set; }
        public string Answer { get; set; } = string.Empty;

        public class AnswerToHelpRequestCommandValidator :
AbstractValidator<AnswerToHelpRequestCommand>
        {
            public AnswerToHelpRequestCommandValidator()
            {
                RuleFor(x => x.Id).NotEmpty().WithMessage("Id is required");
                RuleFor(x => x.Answer).NotEmpty().WithMessage("Answer is required")
                    .MinimumLength(50)
                    .MaximumLength(5000);
            }
        }

        public class AnswerToHelpRequestCommandHandler :
IRequestHandler<AnswerToHelpRequestCommand, HelpRequestDto>
        {
            private readonly IWebApiDbContext _context;
            private readonly IMapper _mapper;

            public AnswerToHelpRequestCommandHandler(IWebApiDbContext context, IMapper
mapper)
            {
```

```
            _context = context;
            _mapper = mapper;
        }

        public async Task<HelpRequestDto> Handle(AnswerToHelpRequestCommand request,
            CancellationToken cancellationToken)
        {
            var helpRequest = await _context.HelpRequests.Where(a => a.Id ==
request.Id).FirstOrDefaultAsync();
            if (helpRequest is null)
                throw new NotFoundException(nameof(HelpRequestEntity), request.Id);
            helpRequest.Answer = request.Answer;
            helpRequest.Status = HelpRequestStatus.Processed;
            await _context.SaveChangesAsync();
            return _mapper.Map<HelpRequestDto>(helpRequest);
        }
    }
  }
}
```

CreateHelpRequestCommand.cs

```
using AutoMapper;
using FluentValidation;
using MediatR;
using WebApiApplication.Context;
using WebApiCore.Models;

namespace WebApiApplication.Features.HelpRequestFeatures.Commands;

public class CreateHelpRequestCommand : IRequest<Guid>
{
  public string? Title { get; set; }
  public string? Description { get; set; }
  public string? Username { get; set; }
  public string? UserEmail { get; set; }
  public Guid? UserId { get; set; }
  public bool? EmailConfirmed { get; set; }

  public class CreateHelpRequestCommandValidator :
AbstractValidator<CreateHelpRequestCommand>
  {
    public CreateHelpRequestCommandValidator()
    {
      RuleFor(x => x.Title).NotEmpty().WithMessage("Title is required");
      RuleFor(x => x.Description).NotEmpty().WithMessage("Description is required");
      RuleFor(x => x.Username).NotEmpty().WithMessage("Username is required");
      RuleFor(x => x.UserEmail).NotEmpty().WithMessage("UserEmail is required");
      RuleFor(x => x.EmailConfirmed).NotEmpty().WithMessage("EmailConfirmed is
required")
        .Equal(true).WithMessage("Email must be confirmed");
      RuleFor(x => x.UserId).NotEmpty().WithMessage("UserId is required");
    }
```

```
    }

    public class CreateHelpRequestCommandHandler : IRequestHandler<CreateHelpRequestCommand, Guid>
    {
        private readonly IWebApiDbContext _context;
        private readonly IMapper _mapper;

        public CreateHelpRequestCommandHandler(IWebApiDbContext context, IMapper mapper)
        {
            _context = context;
            _mapper = mapper;
        }

        public async Task<Guid> Handle(CreateHelpRequestCommand command, CancellationToken cancellationToken)
        {
            var helpRequest = _mapper.Map<HelpRequestEntity>(command);
            _context.HelpRequests.Add(helpRequest);
            await _context.SaveChangesAsync();
            return helpRequest.Id;
        }
    }
}
```

DeleteHelpRequestCommand.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using MediatR;
using Microsoft.EntityFrameworkCore;
using WebApiApplication.Context;
using WebApiApplication.Features.HelpRequestFeatures.Dto;
using WebApiApplication.Shared.Exceptions;
using WebApiCore.Models;

namespace WebApiApplication.Features.HelpRequestFeatures.Commands
{
    public class DeleteHelpRequestCommand : IRequest<HelpRequestDto>
    {
        public Guid Id { get; set; }
        public class DeleteHelpRequestCommandHandler : IRequestHandler<DeleteHelpRequestCommand, HelpRequestDto>
        {
            private readonly IWebApiDbContext _context;
            private readonly IMapper _mapper;
            public DeleteHelpRequestCommandHandler(IWebApiDbContext context, IMapper mapper){
                _context = context;
                _mapper = mapper;
```

```
        }
        public async Task<HelpRequestDto> Handle(DeleteHelpRequestCommand request, CancellationToken cancellationToken)
        {
            var helpRequest = await _context.HelpRequests.Where(a => a.Id == request.Id).FirstOrDefaultAsync();
            if(helpRequest is null)
                throw new NotFoundException(nameof(HelpRequestEntity), request.Id);
            helpRequest.Status = HelpRequestStatus.Removed;
            await _context.SaveChangesAsync();
            return _mapper.Map<HelpRequestDto>(helpRequest);
        }
    }
  }
}
```

GetHelpRequestCountByFilterQuery.cs

```
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using WebApiApplication.Context;

namespace WebApiApplication.Features.HelpRequestFeatures.Queries;

public class GetHelpRequestCountByFilterQuery : IRequest<int>
{
  public string? Filter { get; set; } = default!;
  public Guid? UserId { get; set; }

  public class GetHelpRequestCountByFilterValidator : AbstractValidator<GetHelpRequestCountByFilterQuery>
  {
    public GetHelpRequestCountByFilterValidator()
    {
      RuleFor(x => x.Filter).NotEmpty().Must(filter => !string.IsNullOrWhiteSpace(filter));
      RuleFor(x => x.UserId).NotEmpty();
    }
  }

  public class GetHelpRequestCountByFilterHandler : IRequestHandler<GetHelpRequestCountByFilterQuery, int>
  {
    private readonly IWebApiDbContext _context;

    public GetHelpRequestCountByFilterHandler(IWebApiDbContext context)
    {
      _context = context;
    }

    public async Task<int> Handle(GetHelpRequestCountByFilterQuery request, CancellationToken cancellationToken)
    {
```

```
            var helpRequestCount = _context.HelpRequests
                .Where(x => x.UserId == request.UserId)
                .Where(x => x.Title.Contains(request.Filter!)
                        || x.Description.Contains(request.Filter!)).AsNoTracking()
                .CountAsync(cancellationToken: cancellationToken);
            return await helpRequestCount;
        }
    }
}
```

GetHelpRequestCountForUserQuery.cs

```
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using WebApiApplication.Context;

namespace WebApiApplication.Features.HelpRequestFeatures.Queries;

public class GetHelpRequestCountForUserQuery : IRequest<int>
{
    public Guid? UserId { get; set; }

    public class GetHelpRequestCountForUserQueryValidator : AbstractValidator<GetHelpRequestCountForUserQuery>
    {
        public GetHelpRequestCountForUserQueryValidator()
        {
            RuleFor(x => x.UserId).NotEmpty();
        }
    }

    public class GetHelpRequestCountForUserQueryHandler : IRequestHandler<GetHelpRequestCountForUserQuery, int>
    {
        private readonly IWebApiDbContext _context;

        public GetHelpRequestCountForUserQueryHandler(IWebApiDbContext context)
        {
            _context = context;
        }

        public async Task<int> Handle(GetHelpRequestCountForUserQuery request, CancellationToken cancellationToken)
        {
            return await _context.HelpRequests.AsNoTracking()
                .CountAsync(x => x.UserId == request.UserId, cancellationToken: cancellationToken);
        }
    }
}
```

GetHelpRequestCountQuery.cs

```
using FluentValidation;
```

```
using MediatR;
using Microsoft.EntityFrameworkCore;
using WebApiApplication.Context;
using WebApiCore.Models;

namespace WebApiApplication.Features.HelpRequestFeatures.Queries
{
   public class GetHelpRequestCountQuery : IRequest<int>
   {
      public string? Status {get;set;}
      public class GetHelpRequestCountQueryValidator : AbstractValidator<GetHelpRequestCountQuery>{
         public GetHelpRequestCountQueryValidator()
         {
            RuleFor(x=>x.Status).Must(x=>
            x is null
            || x.ToUpper() == HelpRequestStatus.New.ToString().ToUpper()
            || x.ToUpper() == HelpRequestStatus.Processed.ToString().ToUpper()
            || x.ToUpper() == HelpRequestStatus.Closed.ToString().ToUpper()
            || x.ToUpper() == HelpRequestStatus.Removed.ToString().ToUpper())
            .WithMessage("Status must be null or one of the following: New, Processed, Closed, Removed");
         }
      }
      public class GetHelpRequestCountQueryHandler : IRequestHandler<GetHelpRequestCountQuery, int>
      {
         private readonly IWebApiDbContext _context;

         public GetHelpRequestCountQueryHandler(IWebApiDbContext context)
         {
            _context = context;
         }
         public async Task<int> Handle(GetHelpRequestCountQuery request, CancellationToken cancellationToken)
         {
            if(request.Status is null)
               return await _context.HelpRequests.AsNoTracking().CountAsync(cancellationToken);
            else{
               var status = Enum.Parse<HelpRequestStatus>(request.Status, true);
               return await _context.HelpRequests.Where(x => x.Status == status).AsNoTracking().CountAsync(cancellationToken);
            }
         }
      }
   }
}
```

GetHelpRequestForUserByPageQuery.cs

```
using AutoMapper;
using AutoMapper.QueryableExtensions;
```

```
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using WebApiApplication.Context;
using WebApiApplication.Features.HelpRequestFeatures.Dto;

namespace WebApiApplication.Features.HelpRequestFeatures.Queries;

public class GetHelpRequestForUserByPageQuery : IRequest<IEnumerable<HelpRequestDto>>
{
   public int? Page { get; set; }
   public int? PageSize { get; set; }
   public Guid? UserId { get; set; }

   public class GetHelpRequestForUserByPageQueryValidator :
AbstractValidator<GetHelpRequestForUserByPageQuery>
   {
      public GetHelpRequestForUserByPageQueryValidator()
      {
         RuleFor(x => x.Page).NotEmpty();
         RuleFor(x => x.PageSize).NotEmpty();
         RuleFor(x => x.UserId).NotEmpty();
      }
   }

   public class
      GetHelpRequestForUserByPageQueryHandler :
IRequestHandler<GetHelpRequestForUserByPageQuery,
         IEnumerable<HelpRequestDto>>
   {
      private readonly IWebApiDbContext _context;
      private readonly IMapper _mapper;

      public GetHelpRequestForUserByPageQueryHandler(IWebApiDbContext context, IMapper
mapper)
      {
         _context = context;
         _mapper = mapper;
      }

      public async Task<IEnumerable<HelpRequestDto>>
Handle(GetHelpRequestForUserByPageQuery request,
         CancellationToken cancellationToken)
      {
         if (request.Page < 1) request.Page = 1;
         if (request.PageSize < 1) request.PageSize = 10;
         var helpRequests = await _context.HelpRequests
            .Where(x => x.UserId == request.UserId)
            .OrderByDescending(x => x.CreatedAt)
            .Skip((request.Page!.Value - 1) * request.PageSize!.Value)
            .Take(request.PageSize.Value)
            .AsNoTracking()
```

```
            .ProjectTo<HelpRequestDto>(_mapper.ConfigurationProvider)
            .ToListAsync(cancellationToken);
        return helpRequests;
    }
  }
}
```

GetHelpRequestsByFilterQuery.cs

```
using AutoMapper;
using AutoMapper.QueryableExtensions;
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using WebApiApplication.Context;
using WebApiApplication.Features.HelpRequestFeatures.Dto;

namespace WebApiApplication.Features.HelpRequestFeatures.Queries;

public class GetHelpRequestsByFilterQuery : IRequest<IEnumerable<HelpRequestDto>>
{
  public string? Filter { get; set; }
  public Guid? UserId { get; set; }
  public int? Page { get; set; }
  public int? PageSize { get; set; }

  public class GetHelpRequestsByFilterQueryValidator :
AbstractValidator<GetHelpRequestsByFilterQuery>
  {
    public GetHelpRequestsByFilterQueryValidator()
    {
      RuleFor(x => x.Filter).NotEmpty().Must(filter => !string.IsNullOrWhiteSpace(filter));
      RuleFor(x => x.UserId).NotEmpty();
      RuleFor(x => x.Page).NotEmpty();
      RuleFor(x => x.PageSize).NotEmpty();
    }
  }

  public class
    GetHelpRequestsByFilterQueryHandler :
IRequestHandler<GetHelpRequestsByFilterQuery, IEnumerable<HelpRequestDto>>
  {
    private readonly IWebApiDbContext _context;
    private readonly IMapper _mapper;

    public GetHelpRequestsByFilterQueryHandler(IWebApiDbContext context, IMapper
mapper)
    {
      _context = context;
      _mapper = mapper;
    }
```

```
    public async Task<IEnumerable<HelpRequestDto>>
Handle(GetHelpRequestsByFilterQuery request,
        CancellationToken cancellationToken)
    {
        if (request.Page < 1)
            request.Page = 1;
        if (request.PageSize < 1)
            request.PageSize = 10;
        var helpRequests = await _context.HelpRequests
            .Where(x => x.UserId == request.UserId)
            .Where(x => x.Title.Contains(request.Filter!)
                    || x.Description.Contains(request.Filter!))
            .Skip((request.Page!.Value - 1) * request.PageSize!.Value)
            .Take(request.PageSize.Value)
            .AsNoTracking()
            .ProjectTo<HelpRequestDto>(_mapper.ConfigurationProvider)
            .ToListAsync(cancellationToken);
        return helpRequests;
    }
  }
}
```

GetHelpRequestsQuery.cs

```
using AutoMapper;
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using WebApiApplication.Context;
using WebApiApplication.Features.HelpRequestFeatures.Dto;
using WebApiCore.Models;

namespace WebApiApplication.Features.HelpRequestFeatures.Queries
{
  public class GetHelpRequestsQuery : IRequest<IEnumerable<HelpRequestDto>>
  {
    public string? Status { get; set; }

    public class GetHelpRequestsQueryValidator : AbstractValidator<GetHelpRequestsQuery>
    {
      public GetHelpRequestsQueryValidator()
      {
        RuleFor(x => x.Status).Must(x =>
            x is null
            || x.ToUpper() == HelpRequestStatus.New.ToString().ToUpper()
            || x.ToUpper() == HelpRequestStatus.Processed.ToString().ToUpper()
            || x.ToUpper() == HelpRequestStatus.Closed.ToString().ToUpper()
            || x.ToUpper() == HelpRequestStatus.Removed.ToString().ToUpper())
          .WithMessage("Status must be null or one of the following: New, Processed, Closed,
Removed");
      }
    }
```

```
    public class GetHelpRequestsQueryHandler : IRequestHandler<GetHelpRequestsQuery,
IEnumerable<HelpRequestDto>>
    {
        private readonly IWebApiDbContext _context;
        private readonly IMapper _mapper;

        public GetHelpRequestsQueryHandler(IWebApiDbContext context, IMapper mapper)
        {
            _context = context;
            _mapper = mapper;
        }

        public async Task<IEnumerable<HelpRequestDto>> Handle(GetHelpRequestsQuery
request,
            CancellationToken cancellationToken)
        {
            IEnumerable<HelpRequestEntity> helpRequests;
            if (request.Status is null)
            {
                helpRequests = await _context.HelpRequests
                    .AsNoTracking()
                    .ToListAsync(cancellationToken);
            }
            else
            {
                var status = Enum.Parse<HelpRequestStatus>(request.Status, true);
                helpRequests = await _context.HelpRequests
                    .Where(x => x.Status == status)
                    .AsNoTracking()
                    .ToListAsync(cancellationToken);
            }

            return _mapper.Map<IEnumerable<HelpRequestDto>>(helpRequests);
        }
    }
  }
}
```